Komponentenorientierende Methode zur Simulation von Mehrkörpersystemen

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur
(Dr.-Ing.)

von: **Vlasenko Dmitry**

geb. am 14.09.1977 in Prokopievsk / Russland

genehmigt durch die Fakultät für Maschinenbau
der Otto-von-Guericke-Universität Magdeburg

Gutachter:

Prof. Dr.-Ing. Roland Kasper

Prof. Dr.-Ing. Lutz Sperling

Dr.-Ing. Vladimir Bykov

Tag der Einreichung: 29.09.2005
Promotionskolloquium am: 26.01.2006

Component-Oriented Method for Simulation of Multibody Dynamics

Accepted dissertation

by the Faculty of Mechanical Engineering at the

Otto von Guericke University Magdeburg in fulfilment of the requirements for the academic degree of

Doctor of Engineering (Dr.-Ing.)

by: **Vlasenko Dmitry**

born in14.09.1977 in Prokopievsk / Russia

Advisor:	Prof. Dr.-Ing. Roland Kasper
Co-advisors:	Prof. Dr.-Ing. Lutz Sperling, Dr.-Ing. Vladimir Bykov

Submission date:	29.09.2005
Defence date:	26.01.2006

Acknowledgements

This doctoral thesis is my performed task as a doctoral student at the Institute of Mechatronics and Drives (IMAT) at Otto-von-Guericke-University Magdeburg.

Firstly, I would like to give my particular thanks to my first advisor Prof. Dr.-Ing. Roland Kasper, Managing Director of IMAT for his continual encouragement and patient guidance throughout the course of this work.

I would like to thank my mother Anna for her patience and unlimited love. My deepest thanks go to my wife, Marina. Her love and support through all the emotional ups and downs of the PhD has been unflagging.

I would also like to thank my co-advisors Prof. Dr.-Ing. Lutz Sperling and Dr.-Ing. Vladimir Bykov. Their helpful comments during the research and writing of this thesis are greatly appreciated.

Abstract

The development of a tool for simulation of constrained multibody dynamics is a sophisticated problem. There are a lot of conditions that the simulating tool should satisfy: numerical efficiency, stability, distributivity, flexibility, interaction with other tools, distributed development, etc.

Trying to answer the requirements, we developed and implemented the method of distributed simulation of mechanical systems. Unlike a huge number of other methods, we keep the block-module concept during simulation. The main advantages of our approach are separate testing of subsystems, encapsulation of critical effects inside of subsystems and distributed simulation of subsystems.

It is an exact, non-iterative algorithm that is applicable to mechanisms with any joint type and any topology, including branches and kinematic loops. The technique can be implemented for various systems of connected bodies with variable number of degrees of freedom such as systems with coulomb frictions.

Complexity of the simulation of good-partitioned systems requires $O(n)$ floating point operations, that is comparable with the fastest available algorithms. The combination of generalized and absolute coordinates significantly increases the method's efficiency.

The object-oriented implementation of the algorithm significantly reduces the cost and development time of modelling. The tests use a car system with a closed-loop structure as one example and a spatial manipulator as another. Both models are performed using an object-oriented approach, with several levels of hierarchy. Numerical simulation shows the stability of the method. Drift is constant and is limited to the order of the computation accuracy.

For the validation of the simulations results we have built up the same models in Dymola and Simpack software. The comparison shows that the dynamics of the models was calculated correctly.

Table of Contents

Mathematical Notation

s – Scalars, italic, Tymes New Roman font

$\underline{q}$ - Function, underline, italic, Tymes New Roman font

$\mathbf{v}$ – Vector, bold lower case, Tymes New Roman font

$\mathbf{A}$ – Matrix, bold upper case, Tymes New Roman font

$\boldsymbol{I}$ – Array, bold italic upper case, Tymes New Roman font

$Ground$ – Objects, italic, Tymes New Roman font

1 Introduction

The dynamics of multibody systems, such as motion of robotic manipulators, vehicle systems and spacecrafts, is becoming increasingly important in engineering, especially in mechatronics. A computer simulation of such multibody systems requires a concerted integration involving several computational aspects [HAU 90, SHL 90, SHL 93]. These include selection of a data structure for the system's configuration, computerized generation of governing equations of motion, incorporation of constraint conditions and implementation of suitable solution algorithms. Basic methods for multibody system simulations are provided by the disciplines of dynamics (the multibody formulations), numerical mathematics and computer science [EIC 93].

Let us briefly review the problems of simulation tools. At first we show the most popular theoretical methods of simulation and then we review some important implementation's aspects.

In this thesis we discuss only the simulation of holonomic systems, though the method can be easily generalized for the simulation of nonholonomic systems.

1.1 Algorithms of Simulation

The principal problem associated with the simulation of constrained mechanical systems is *forward dynamics*. Given the time-histories of actuated joint torques and forces, we need to compute their time-histories of the joint coordinates, velocities and accelerations. In this case, the solution is obtained in a two-stage process. In the first stage, the equations of motion are solved algebraically to determine the accelerations. In the second stage, the underlying ordinary differential equations (ODE) are integrated to obtain all the joint-coordinate time histories.

Methods for formulation of equations of motion fall into two main categories: a) Euler-Lagrange and b) Newton-Euler formulations. Typically, *Euler-Lagrange* formulations use joint-based relative coordinates as configuration-space variables; these

formulations are generally not well suited for a recursive formulation. However, they are popular within the robotics community, since they use joint-based relative coordinates, which form a minimal-set for serial manipulators and have a direct technical meaning in robotics. *Newton-Euler* approaches typically use Cartesian variables as configuration-space variables. They admit recursive formulations by first developing equations of motion for each single body; these equations are then assembled to obtain the model of the entire system.

In subsequent discussions we will focus on the development of equations of motion of constrained mechanical systems with loops.

1.1.1 Recursive Newton-Euler Formulations

Dynamics equations based on classic Lagrange approaches are of the order $O(n^4)$ [FEA 87], which means that the number of floating point operations grow with the fourth power of the number of bodies n in the system. Many variants of fast and readily-implementable recursive algorithms have been formulated within the last two decades, principally within the robotics community.

The earliest $O(n)$ algorithm for *forward dynamics* was developed by Vereshchagin [VER 74] who used a recursive formulation to evaluate the Gibbs-Appel form of the equations of motion and is applicable to unbranched chains with revolute and prismatic joints. Next, Armstrong [ARM 79] developed an $O(n)$ algorithm for mechanisms with spherical joints. Later, Walker and Orin [WAL 82] developed an efficient recursive forward dynamics algorithm. This method is commonly referred to as the *composite-rigid-body algorithm* (CRBA). This algorithm needed to solve a linear system of equations whose dimension grows with the number of rigid bodies. Since methods to solve a linear system of n equations in the n unknowns are $O(n^3)$, this algorithm is also $O(n^3)$. However, for small n, the first-order terms dominate the computation, so that the algorithm is quite efficient. So far, the CRBA is perhaps the most efficient general-purpose algorithm for serial manipulators with $n < 10$, which includes most practical cases.

Next, Featherstone [FEA 83] developed what he called the *articulated-body algorithm* (ABA), which was followed by a more elaborate and faster model [FEA 87]. The

computational complexity of ABA is $O(n)$ and is more efficient than CRBA for $n > 9$. Further gains have been made in efficiency over the years [BRA 86, MML 95].

In multi-loop mechanisms the joint variables are no longer independent, since they are subject to loop-closure constraints, which are usually nonlinear. The existing literature on recursive algorithms applied to multi-loop mechanisms almost always uses a non-minimal set of generalized coordinates [BAE 87, CHL 90a, STE 96, BAE 99, FEA 99]. The most common method for dealing with kinematics is to cut the loop, introduce Lagrange multipliers to substitute for the cut joints and use a recursive scheme for the open-chain system to obtain a recursive algorithm. However, the methods have strong problems with stability.

1.1.2 Non-Recursive Newton-Lagrange Formulations

The dynamics of constrained mechanical systems with *closed loops* using a Newton-Lagrange approach is traditionally obtained by cutting the closed loops to obtain various open loops, also known as reduced systems, and then writing a system of ODEs for the corresponding chains in their corresponding generalized coordinates [FEA 87]. The solution to these is required to satisfy additional algebraic equations, which typically are constraint equations required to close the cut-open loops. A *Lagrange multiplier* term is introduced to represent the forces in the direction of the constraint violation. The resulting formulation, often referred to as a *descriptor form,* yields an often simpler, even though larger, system of index-3 differential algebraic equations (DAEs) as follows:

$$\dot{\mathbf{p}} = \mathbf{T}_p \mathbf{w} \tag{1.1}$$

$$\mathbf{M}(\mathbf{p})\dot{\mathbf{w}} = \mathbf{f}(\mathbf{p}, \mathbf{w}) - \mathbf{G}^T(\mathbf{p})\boldsymbol{\lambda} \tag{1.2}$$

$$\mathbf{g}(\mathbf{p}) = 0 \tag{1.3}$$

where

$\mathbf{p}$ is the vector of generalized coordinates,

$\mathbf{w}$ is the vector of generalized velocities,

$\mathbf{M}(\mathbf{p})$ is the mass matrix,

$\mathbf{f}(\mathbf{p},\mathbf{w})$ is the vector of external forces (other than constrain forces),

$\mathbf{g(p)}$ is the vector of holonomic constraints,

$\mathbf{G}(\mathbf{p}) = \frac{\partial \mathbf{g}}{\partial \mathbf{p}} \mathbf{T}_p$ is the product of the constraint Jacobian matrix $\frac{\partial \mathbf{g}}{\partial \mathbf{p}}$ and the transformation matrix $\mathbf{T}_p$,

$\boldsymbol{\lambda}$ is the vector of Lagrange multipliers.

Remark 1.1 *For notational simplicity, we assume that the matrix* $\mathbf{G}$ *is the Jacobian matrix of* $\mathbf{g(p)}$ *and assume the matrix* $\mathbf{T}_p$ *in* (1.1) *is the identity matrix. Our discussion on a general form can be made through minor modifications.*

The solution of a system of index-3 DAEs by direct finite difference discretization is not possible using explicit discretization methods [AHR 98]. Instead, the above system is typically converted to a system of ODEs and expressed in state-space form, which may be integrated using standard numerical code. Below we discuss the most popular conversion's methods.

1.1.2.1 Direct Elimination

The surplus variables are eliminated directly, using the equations of constraints to explicitly reduce index-3 DAE to an ODE in a minimal set of generalized coordinates (conversion into Lagrange's equations of the second kind). This is also referred to as a *closed-form solution* of the constraint equations. The resulting minimal order ODE is stable and can then be integrated. However, such a reduction cannot be done in general, and even when it can, the differential equations obtained, are typically complicated [KEC 97].

1.1.2.2 Lagrange-Multiplier Approximation-Penalty Formulation

In this approach the loop-closure constraints are *relaxed* and replaced by virtual springs and dampers [WAN 00]. It looks like a form of penalty formulation [GAR 94], which incorporates the constraint equations as a dynamical system penalized by a large factor. The Lagrange multipliers are *estimated* using a compliance-based force-law. The latter is based on the extent of constraint violation and assumed spring

stiffness; the force is then eliminated from the list of $n+c$ unknowns, leaving behind a system of $2n$ first-order ODEs, where c is the size of $\mathbf{g(p)}$. The choice of parameters of virtual springs and dampers is a sophisticated problem. It is important to note that penalty approaches only approximate the true constraint forces and can create unanticipated problems.

1.1.2.3 Lagrange-Multiplier Elimination

A very popular approach in practice is to differentiate the constraints twice, obtaining at each time t an algebraic system for the accelerations and the Lagrange multipliers. Thus, differentiating the position constraints (1.3) once, we obtain the constraint equations on velocity level

$$\mathbf{0} = \dot{\mathbf{g}} = \mathbf{G(p)w} \tag{1.4}$$

and a further differentiation with respect to time results in the constraint equations on acceleration level

$$\mathbf{0} = \ddot{\mathbf{g}} = \mathbf{G(p)\dot{w}} + \dot{\mathbf{G}}\mathbf{(p,w)w} \tag{1.5}$$

Remark 1.2 *Throughout the thesis we will refer to* (1.3) *as the position constraints, to* (1.4) *as the velocity constraints and to* (1.5) *as the acceleration constraints, although of course these are all just different forms of the original constraints which are given on the generalized position coordinates.*

Combining (1.1), (1.2) with (1.5), we get:

$$\begin{pmatrix} \mathbf{M} & \mathbf{G}^T \\ \mathbf{G} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \dot{\mathbf{w}} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \varphi \end{pmatrix} \tag{1.6}$$

where $\varphi = -\dot{\mathbf{G}}\mathbf{(p,w)} \cdot \mathbf{w}$.

This allows elimination of $\boldsymbol{\lambda}$ in terms of the accelerations $\dot{\mathbf{w}}$, obtaining an ODE system for $\mathbf{w}$ and $\mathbf{p}$:

$$\begin{aligned} \dot{\mathbf{p}} &= \mathbf{w} \\ \mathbf{M(p)\dot{w}} &= \tilde{\mathbf{f}}\mathbf{(p,w)} \end{aligned} \tag{1.7}$$

where $\tilde{\mathbf{f}}(\mathbf{p},\mathbf{w}) = \mathbf{f} - \mathbf{G}^T\left(\mathbf{G}\mathbf{M}^{-1}\mathbf{G}^T\right)^{-1}\left(\mathbf{G}\mathbf{M}^{-1}\mathbf{f} - \varphi\right)$

This system may be integrated using standard codes.

Remark 1.3 *Note that, in principle, the index-reduced system* (1.6) *or* (1.7) *needs more initial conditions than the original system* (1.1) *to specify a unique solution. We assume, however, that consistent initial conditions* (*see, e.g. [BRE 89]*) *for the generalized position and velocity coordinates are provided.*

However, there is a disadvantage to integrate (1.7) or (1.6) numerically. The position and velocity constraints (1.3) and (1.4) are no longer satisfied exactly - there is a *drift* off the constraints, which results in an error of motion and velocity for longer simulations. Moreover, though, the drift magnitude as well as the error in generalized positions and velocities grows with time t - at worst quadratically [BAU 72, ALI 92, AHR 93]. This is not because of the numerical method used to integrate (1.7) but because the system (1.7) or (1.6) itself is mildly unstable. Below we review the stabilization methods that help to solve the problem.

1.1.2.4 Baumgarte's technique

Using Baumgarte's technique [BAU 72], we consider the index-1 DAE (1.6) or the corresponding ODE (1.7) obtained by eliminating the Lagrange multipliers, but now φ is defined by

$$\varphi = -\dot{\mathbf{G}}(\mathbf{p},\mathbf{w})\mathbf{w} - \alpha_1\dot{\mathbf{g}}(\mathbf{p},\mathbf{w}) - \alpha_0\mathbf{g}(\mathbf{p}) \tag{1.8}$$

where the parameters α_j are chosen so that the roots of the polynomial

$$\sigma(\tau) = \tau^2 + \alpha_1\tau + \alpha_0 \tag{1.9}$$

both have negative real parts. For instance, one may choose

$$\sigma(\tau) = (\tau + \gamma)^2 \tag{1.10}$$

for some $\gamma > 0$. The effect of this is to replace (1.5) by

$$\mathbf{0} = \ddot{\mathbf{g}} + 2\gamma\dot{\mathbf{g}} + \gamma^2\mathbf{g} \tag{1.11}$$

The apparent conceptual simplicity of Baumgarte stabilization technique and the fact that it essentially replaces the index-3 DAE (1.1) - (1.3) by an ODE formulation must be considered a major reason for its popularity in engineering applications.

The disadvantage of the method is the practical choice of parameters (e.g. γ in (1.11)) to make the stabilization robust. The optimal γ depends on both the discretization step size h and the discretization method [BRE 89]. Nowadays there is no sufficient algorithm for calculation of γ.

1.1.2.5 Projected invariants methods

Another technique is maintaining more constraints by introducing additional multipliers μ [AHR 93, GEA 81, GEA 85]. By using this technique, DAE (1.1)-(1.4) can be reformulated as [GEA 85]

$$\begin{aligned} \dot{\mathbf{p}} &= \mathbf{w} + \mathbf{G}^T(\mathbf{p})\boldsymbol{\mu} \\ \mathbf{M}\dot{\mathbf{w}} &= \mathbf{f}(\mathbf{p},\mathbf{w}) - \mathbf{G}^T(\mathbf{p})\boldsymbol{\lambda} \\ \mathbf{0} &= \mathbf{G}(\mathbf{p})\mathbf{w} \\ \mathbf{0} &= \mathbf{g}(\mathbf{q}) \end{aligned} \tag{1.12}$$

The system (1.12) is an index-2 DAE for variables ($\mathbf{p}$, $\mathbf{w}$, $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$). The exact solution for $\boldsymbol{\mu}$ is $\boldsymbol{\mu} \equiv \mathbf{0}$ so that (1.12) and (1.7) will share the same solutions for ($\mathbf{p}$, $\mathbf{w}$, $\boldsymbol{\lambda}$). As the numerical solution of (1.12) satisfies both the position constraint (1.3) and the velocity constraint (1.4) the method has no drift problem. But the computation of (1.12) could be expensive as implicit schemes have to be used for the even larger dimension (1.12).

1.1.2.6 Dynamic Projection onto the Tangent Space

Describing vectors and matrices, we show in square brackets their size. Vectors (column vectors) are simply matrices with a single column.

These methods seek to take the equations of motion into the selected constraints manifold. Let $\mathbf{S}(\mathbf{p})$ be a [*n, n-c*] full-rank matrix whose column space lies in the nullspace of $\mathbf{G}(\mathbf{p})$, i.e. $\mathbf{G}(\mathbf{p})\mathbf{S}(\mathbf{p}) = \mathbf{0}$. All *feasible* dependent velocities $\mathbf{w}$ belong to the space, which is spanned by the columns of $\mathbf{S}(\mathbf{p})$:

$$\mathbf{w}=\mathbf{S}(\mathbf{p})\mathbf{u}(\mathbf{t})$$

where $\mathbf{u(t)}$ is n-c dimensional vector of independent velocities.

Using the matrix $\mathbf{S}$ we could obtain from DAE (1.1) - (1.3) the ODE:

$$\dot{\mathbf{u}} = \mathbf{f}(\mathbf{p}, \mathbf{w}, \mathbf{u})$$

that can be integrated with suitable ODE solvers.

A family of choices exists for the selection of dependent and independent velocities [SHA 01, GAR 94].

1.1.2.7 Post-Stabilizations Method

The post-stabilisation method [AHR 95] relates to coordinate projection methods. Nowadays, this is one of the most effective and convenient methods for the simulation of constrained mechanical systems.

The position and velocity constraints together form an invariant set Θ of ODE (1.7), given by

$$\mathbf{0} = \mathbf{h}(\mathbf{p}, \mathbf{w}) = \begin{pmatrix} \mathbf{g(p)} \\ \mathbf{G(p)w} \end{pmatrix} \tag{1.13}$$

While the simulation on each time step we perform the following two-stage subroutine:

1. Using a favourite ODE integration scheme (e.g. Runge-Kutta or multistep) we obtain from (1.12) the values of $\tilde{\mathbf{p}}_{k+1}$, $\tilde{\mathbf{w}}_{k+1}$ on the new time step.

2. Stabilize:

$$\begin{pmatrix} \mathbf{p}_{k+1} \\ \mathbf{w}_{k+1} \end{pmatrix} = \begin{pmatrix} \tilde{\mathbf{p}}_{k+1} \\ \tilde{\mathbf{w}}_{k+1} \end{pmatrix} - \mathbf{F}(\tilde{\mathbf{p}}_{k+1}, \tilde{\mathbf{w}}_{k+1})\mathbf{h}(\tilde{\mathbf{p}}_{k+1}, \tilde{\mathbf{w}}_{k+1}) \tag{1.14}$$

   where

$$\mathbf{F} = \mathbf{Q}(\mathbf{HQ})^{-1}$$
$$\mathbf{H} = \begin{pmatrix} \frac{\partial \mathbf{h}}{\partial \mathbf{p}} & \frac{\partial \mathbf{h}}{\partial \mathbf{w}} \end{pmatrix} = \begin{pmatrix} \mathbf{G} & \mathbf{0} \\ \dot{\mathbf{G}} & \mathbf{G} \end{pmatrix} \tag{1.15}$$

with $\mathbf{Q}(\mathbf{p},\mathbf{w})$ smooth such that $\mathbf{HQ}$ is nonsingular.

Remark 1.4 *Here we assume that* $\mathbf{H}$ *has a full row rank. In practice, during the simulation we use* a *pseudoinverse formula based on singular value decomposition* [CLI 03].

The post-stabilization guarantees [CHI 95] the asymptotic stability of Θ in the difference equations even when Θ is slightly unstable in the underlying vector field. Therefore the numerical solutions will stay near Θ for all time integration.

In our software we find it most convenient to choose

$$\mathbf{Q} = \begin{pmatrix} \mathbf{G}^{\mathrm{T}} & \mathbf{0} \\ \mathbf{0} & \mathbf{G}^{\mathrm{T}} \end{pmatrix} \tag{1.16}$$

In the case when $\dot{\mathbf{G}}$ does not significantly dominate, we can neglect it and rewrite (1.15) as

$$\mathbf{F}(\mathbf{p},\mathbf{w}) = \mathbf{G}^{T}\left(\mathbf{G}\mathbf{G}^{T}\right)^{-1}\begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \tag{1.17}$$

where $\mathbf{I}$ is the identity matrix.

In next chapters we consider some simulation examples that show the stability of the technique.

1.1.3 Distributed Forward Dynamic Simulation

The simulation process involves the time-discretized numerical solution of an initial-value problem, using a variety of numerical time-stepping schemes. In particular, the *numerical stiffness* of the underlying coupled differential-algebraic equations necessitates a large number of small time-steps in order to ensure a prescribed accuracy. Hence, while *real-time* and *interactive* simulations of complex systems are desirable from a design view point, they tend to be difficult to achieve for large multi-

body systems with multiple links and many kinematic loops using conventional processing paradigms. One method to achieve speed-ups in such computations and to satisfy real-time constraints is to *distribute* the computational load onto several processors running in parallel. Henrich and Honiger [HEN 97] gave a brief review and a preliminary classification of the different levels of distribution that have been explored in the context of robotic applications and noted that distribution at all levels may not be possible. Results obtained by distributed algorithms vary depending on the degree of dependency and coupling among the equations. While image-processing problems [CHA 90] can be broken down quite well by dividing the image into smaller independent blocks, the problems of simulation of constrained mechanical systems is a strongly coupled problem and the task is not trivially distributable [FUJ 92, ZOY 93].

In what follows, we will discuss some aspects of these levels of distribution as applicable to the simulation of robotic systems, and specifically to closed-loop systems.

1.1.3.1 Constraint-force algorithm

Fijani *et al.* [FUJ 95] are credited for the first distributed forward dynamics algorithm called the *constraint-force algorithm* (CFA) for *serial/parallel* manipulators with $O(\log(n))$ complexity of computation on $O(n)$ processors. An improved form of this, where all restrictions to type of kinematic chains and classes of joints were removed, appeared in [FEA 99]. The algorithm is in full-descriptor form and works by dividing the mechanisms into sub-chains, obtaining a sparse system of linear equations for the unknown inter-body constraint forces. This system is then solved by various iterative parallel methods. The constraint forces are then used to determine state-derivatives that are time-integrated to obtain updated values for the system state. The main disadvantage to this method is the utilization of iterative methods and the use of the full descriptor form, which is not stable.

1.1.3.2 Divide-and-conquer articulated-body algorithm

The *divide-and-conquer articulated-body algorithm* (DCA) [FEA 99a] with $O(\log(n))$ time complexity on $O(n)$ processors is the fastest available algorithm for a computer with a large number of processors and low communication cost.

The method uses a recursive binary assembly of a system, as shown in Fig. 1.1. Each assembly corresponds to the assembly tree, where leaves are bodies and nodes are constrained subsystems. The central idea of the method is that it is possible to construct the equations of motion of each node in the assembly tree from the corresponding equations of its children.

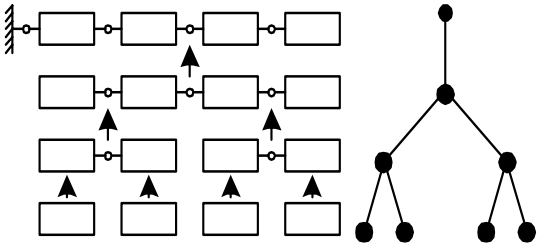

Figure 1.1: Recursive binary assembly of a four-link chain and the corresponding assembly tree

The complete DCA consists of four passes through the virtual processor tree. The first two passes serve to calculate the body positions and velocities. Using the assembly tree we calculate the new values of position and velocity variables of child from the current value of the joint position and velocity variables.

In the third pass we start from the leaves of the assembly tree and work toward the root. Subsystems express the acceleration of their external joints as the linear functions of forces acting in this joints and transform dependency matrices up to tree.

The fourth pass is the back-substitution pass, in which subsystems calculate the acceleration of internal joint from the known forces in external joints.

The approach has several disadvantages. The first is the high communication cost, that limits the method's implementation on general-purpose parallel and multiprocessing systems such as distributed-memory cluster computing machines.

The second drawback is the use of Baumgarte stabilization for closed-loops systems. Above we discussed the problems of the practical use of the stabilization. Also the drift of Baumgarte stabilization is much more than the drift of some other stabilization techniques.

The next problem is the limit on the structure of simulating systems: only one body can be connected with the ground. Obviously, there are many popular multibodies systems (e.g. multilegged robots, vehicles) that do not satisfy this limit.

We note also that the most common way is the construction of a simulating system from subsystems in a hierarchical approach, i.e. we consider the complete system as the highest level of hierarchically connected subsystems. But the assembly tree of the complete model changes after addition of a new subsystem. Thus, we can not partially test the subsystems, but should test the complete model, that significantly increases development costs.

All this drawbacks limits the implementation of DCA method in practical use. Nowadays, DCA method is not implemented in commercial tools. Also, there are no tests of DCA's stability in the case of simulation of closed-loops systems.

In our method we perform a hierarchical calculation of accelerations in the way similar to DCA. But we simulate the model using the same hierarchy disassembly, as was performed by the user while the model's construction. Since the assembly tree of a subsystem does not changes while the changes of the global model's structure, it follows that we can distributivly test the subsystem. For the stability of the method we use the post-stabilization technique that is much more convenient and accurate than Baumgarte's method. All this helps us to avoid the drawbacks of DCA method.

1.1.3.3 Hybrid Direct/Iterative Algorithm

The Hybrid Direct/Iterative Algorithm (HDIA) proposed by Anderson and Duan [AND 00] is an iterative algorithm and works by cutting a rigid-body system into just sufficient separate pieces to allow for full use of all the processors on a given parallel computer.

The equations of the separate pieces are evaluated in parallel, and the results are loaded into a single system-wide matrix equation to calculate the constraint forces acting between the pieces due to the cut joints. This matrix has dimensions that depend on the number of cut joints, rather than the number of bodies, and is typically sparse, enabling parallel iterative solution techniques to be used effectively. Apart from this one matrix equation, the total cost of the rest of the algorithm is $O(\log(n))$. HDIA expresses its equations of motion in minimal coordinates using coordinate-partitioning, which is an advantage. However, again the iterative solution techniques employed are the major draw back.

1.2 Object-Oriented Implementation

1.2.1 Tool requirement

Modelling and simulation are becoming more important since engineers need to analyze increasingly complex mechanical and mechatronic systems. And in many cases the simulation method's parameters i.e. numerical efficiency, stability and distributivity are much less important than the implementation parameters. Trying to choose the appropriate commercial software, we should evaluate the software efficiency.

1.2.1.1 Flexibility

In earlier years, software components of systems had not been designed for reuse, and any modification in design required a substantial re-evaluation, which made such systems almost as expensive as possible, even more expensive than individually designed ones. But recent years have shown an increasing demand for pre-fabricated goods with lots of options that the customer can choose from. The markets for flexible manufacturing depend heavily on the ability of the producer to maximize flexibility, while keeping the cost down and providing as fast a response time as possible on customized orders.

This goes hand in hand with a demand for flexible modelling and simulation tools, whereby hardware components are described by corresponding software modules

that must be combinable in at least the same flexible manner as the hardware components themselves.

1.2.1.2 Usability

One of the most important characteristics of a tool is its usability. The software should minimize the time of simulating model's redesign. Modelling should be much closer to the way an engineer builds a real system, first trying to find standard components like motors, pumps and valves from manufacturers' catalogues with appropriate specifications and interfaces [ELM 01].

The main factors that help reduce both cost and development time of software are:

- **Reusability**. A software design methodology that ensures optimal reusability of software components is the most essential factor in keeping the software development and maintenance cost down.

- **Quick Development**. Typically, the engineer is facing some particular problems. In order to get a clear arrangement of the distinct physical elements the separation of the real structure into the block elements has to be done in a physical- and design-related manner. Thus one obtains several model-blocks, each of them representing the corresponding mechanical subsystem.

- **Abstraction**. Higher abstraction levels at the user interface help to reduce the time of software development as well as debugging. The conceptual distance between the user interface and the final production code needs to be enlarged. Software translators can perform considerably more tasks than they traditionally did.

1.2.1.3 Interaction with other tools

In last few years the importance of mechatronics significantly grows [KAS 04]. The huge numbers of modern machines are complex mechatronic structures consisting of electronic units, electromechanical transformers such as sensors, actors, pure data processing units as controllers and mechanical structures. The popularity of

mechatronic structures grows enormously: hardware, cars, home electronics like clothes washers and video equipment, robots, airplanes etc.

That is why nowadays one of the most important requirements for a mechanical simulation tool is its interaction with electrical and control tools. Today the world's largest automotive companies estimate that 80-90% of future innovations are based on the integration of electronics and information processing in their classical mechanical products.

But special problems appear when coupling several components from different disciplines to one new system and the methodical limits of the used tool are reached, because of the different engineering domains. One possibility is the translation by analogy consideration [KAS 95]. The other way is to couple different simulation tools, but then there is no direct view to the real system components [Lefarth 96]. Every result and modification has to be translated and very often this can only be done by the model developer. It is clear that this is a major source of errors.

The interaction with other tools is one of the most important parameter of simulation software.

1.2.2 Object-Oriented programming

Trying to satisfy all these demands, modern simulation tools use the object-oriented method. One of the principal advantages of object-oriented programming techniques over procedural programming techniques is that they enable programmers to create modules that do not need to be changed when a new type of object is added. A programmer can simply create a new object that inherits many of its features from existing objects. This makes object-oriented programs easier to modify.

One of the most popular simulation tool Dymola is based on the object-oriented modelling paradigm that was originally invented in 1978 by Hilding Elmqvist as part of his Ph.D. dissertation [ELM 78].

The object-oriented modelling paradigm shares many of the properties of object-oriented programming. Its main characteristics can be summarized as follows [CEL 95]:

- **Encapsulation of knowledge**. The modeller must be able to encode all knowledge related to a particular object in a compact fashion in one place with well-defined interface points to the outside.

- **Topological interconnection capability**. The modeller should be able to interconnect objects in a topological fashion, plugging together component models in the same way as an experimenter would plug together a real equipment in a laboratory. This requirement involves that equations describing a subsystem should be independent on equations of a global model.

- **Hierarchical modelling**. The modeller should be able to declare interconnected models as new objects, making them indistinguishable from the outside from the basic equation models. Models can then be built up in a hierarchical fashion.

- **Object instantiation**. The modeller should have the possibility to describe generic object classes, and instantiate actual objects from these class definitions by a mechanism of model invocation.

- **Class inheritance**. A useful feature is class inheritance, since it allows the encapsulation of knowledge even below the level of a physical object. The so encapsulated knowledge can then be distributed through the model by an inheritance mechanism, which ensures that the same knowledge will not have to be encoded several times in different places of the model separately.

- **Generalized Networking Capability**. A useful feature of a modelling environment is the capability to interconnect models through nodes. Nodes are different from regular models (objects) in that they offer a variable number of connections to them. This feature mandates the availability of across and through variables, so that power continuity across the nodes can be guaranteed.

Kasper and W. Koch [KAS 99] introduced a COM based Mechatronic Design Environment. Their technology allows treating arbitrary models, analysis and design methods in a uniform and implementation independent way, by concentration on a set of well-defined interfaces. This allows the reuse of existing software by

connection interfaces on a very efficient level. Actually there exist interfaces to use models generated by Matlab/Simulink and Dymola. Using their approach, it is possible to simulate even very complex mechatronic models.

2 Theoretical Background

In Chapter 1 we showed the advantages of the object-oriented approach. Unfortunately this type of modularization in most cases is given up during the simulation, especially for mechanical systems, because common modelling formulations use access to the complete system to calculate all accelerations needed. But from a practical point of view, there are big advantages of the simulation on the basis of subsystems:

1. Subsystems can be modelled, tested and compiled. Then they can be used in a way similar to software components that encapsulate their internal structure and can be connected via interfaces.

2. Critical effects like coulomb friction, backslash etc. can be encapsulated inside a subsystem.

3. Subsystems are ideal candidates for the partitioning of large systems on multiple processors.

2.1 Main idea of the hierarchical simulation

Fig. 2.1 shows the multibody system S that was built up by a design engineer as a hierarchy of subsystems. The subsystems $S_{1,1}$, $S_{1,2}$, $S_{2,1}$, $S_{2,2}$ of the first level of the hierarchy consist of connected bodies. The subsystems S_1, S_2 of the second level consist of connected subsystems of the first level. The relation between S_1 and $S_{1,1}$, $S_{1,2}$ are called *inheritance*, S_1 is called *a child* of $S_{1,1}$, $S_{1,2}$. Correspondingly, $S_{1,1}$, $S_{1,2}$ are called *parents* of S_1. The system S consists of the connected subsystems S_1, S_2.

A subsystem is called *basic* if it does not include other subsystems, i.e. the subsystem is situated on the first level of the hierarchy (e.g. $S_{1,1}$, $S_{1,2}$, $S_{2,1}$, $S_{2,2}$). If a subsystem consists of several connected subsystems, then this subsystem is called *derived* (e.g. S_1, S_2 and S).

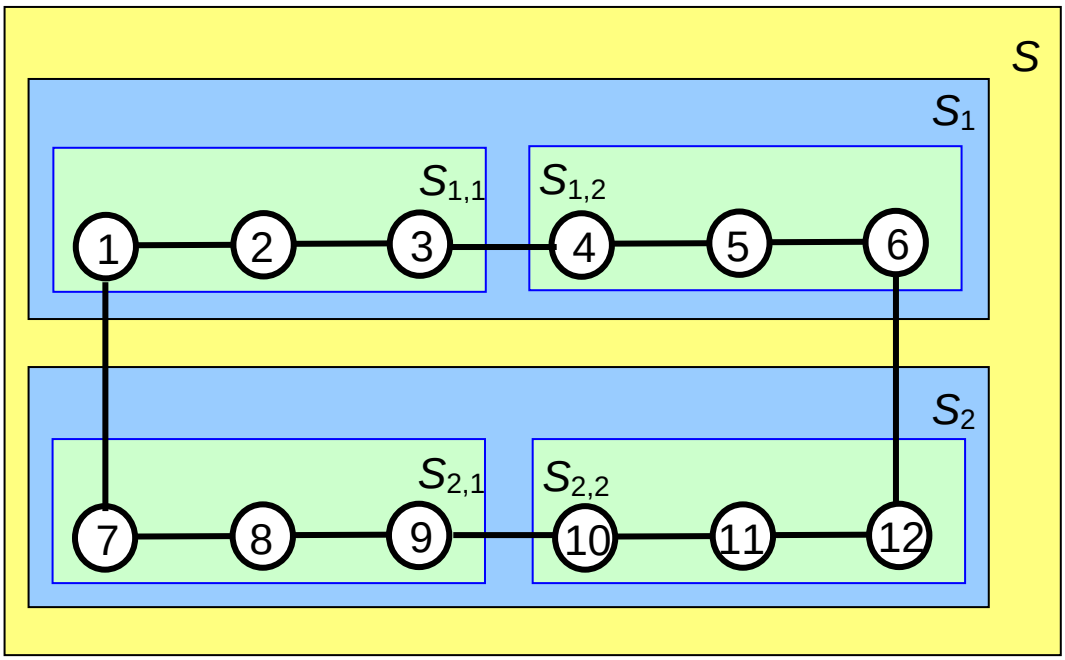

Figure 2.1: Multibody system S

Consider a basic subsystem. A body is called *bordering to the basic subsystem* if it is connected with subsystem's external joints (e.g. *Body 1* and *Body 3* are bordering to the subsystem $S_{1,1}$). All other bodies in the subsystem are called *internal to the basic subsystem* (e.g. *Body 2* is internal to the subsystem $S_{1,1}$).

Consider a derived subsystem. In our method the subsystem needs only the information about bordering bodies of its parents and does not need any information about parents' internal bodies. This approach significantly reduces the size of equations and communication cost. That is why we call a body *internal to the derived subsystem* if it is bordering to one of subsystem's parents and is not connected with subsystem's external joints (e.g. *Body 9* and *Body 10* are internal to the subsystem S_2, but *Body 8* is not internal to the subsystem S_2 because it is internal to $S_{2,1}$). We call a body *bordering to the derived subsystem* if it is bordering to one of subsystem's parents and is connected with subsystem's external joints (e.g. *Body 7* and *Body 12* are bordering to the subsystem S_2). Obviously, S does not have bordering bodies and has four internal bodies: *Body 1*, *Body 6*, *Body 7*, *Body 12*.

During the simulation, on each time step we perform the several operations, shown in Fig. 2.2:

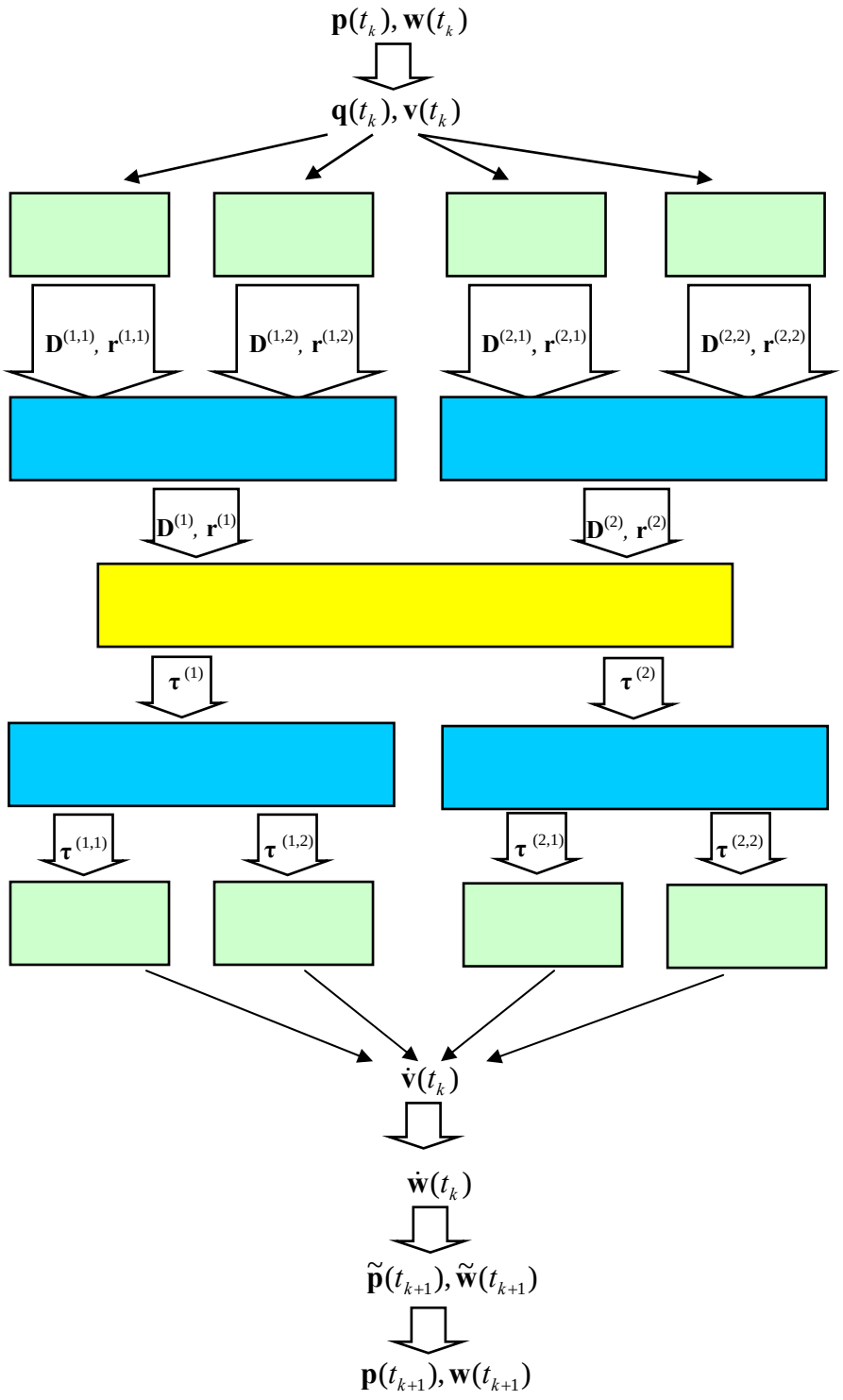

Figure 2.2: Simulation steps

1. **Calculation of absolute coordinates and velocities**. Using current values of the generalized coordinates $\mathbf{p}$ and velocities $\mathbf{w}$, we consequently calculate the absolute coordinates $\mathbf{q}$ and velocities $\mathbf{v}$ of all simulating bodies.

2. **Hierarchical generations of equations of motion.** A subsystem gets from its parents their dependency matrices $\mathbf{D}^{(k)}$ and $\mathbf{r}^{(k)}$:

$$\dot{\mathbf{v}}_{\mathbf{e}}^{(k)} = \mathbf{D}^{(k)}\boldsymbol{\tau}^{(k)} + \mathbf{r}^{(k)}$$

where

$\dot{\mathbf{v}}_{\mathbf{e}}^{(k)}$ is the vector of absolute accelerations of k -th parent's bordering bodies,

$\boldsymbol{\tau}^{(k)}$ is the vector of forces acting in k-th parent's external links.

Using equations of constraints connecting the parents, the subsystem calculates matrices $\mathbf{D}$ and $\mathbf{r}$ and transmits them to its child. Here $\mathbf{D}$ and $\mathbf{r}$ are the dependency matrices:

$$\dot{\mathbf{v}}_{\mathbf{e}} = \mathbf{D}\boldsymbol{\tau} + \mathbf{r}$$

where

$\dot{\mathbf{v}}_{\mathbf{e}}$ is the vector of accelerations of subsystem's bordering bodies,

$\boldsymbol{\tau}$ is the vector of forces acting in subsystem's external links.

3. **Backward hierarchical calculation of absolute accelerations**. A subsystem gets the current values of $\boldsymbol{\tau}$ from its child. Using $\boldsymbol{\tau}$, the subsystem calculates $\dot{\mathbf{v}}_{\mathbf{e}}$. Then for each parent k the subsystem calculates $\boldsymbol{\tau}^{(k)}$ and transmits it to the parent.

   After we reach the lowest level of the hierarchy, the absolute accelerations of all simulating bodies are calculated.

4. **Calculation of generalized accelerations**. From the absolute accelerations $\dot{\mathbf{v}}$ we consequently calculate the current values of the generalized accelerations $\dot{\mathbf{w}}$.

5. **Calculation of generalized coordinates and velocities on the next time step**. Using a favourite ODE integration scheme (e.g. Runge-Kutta or multistep), we obtain the values of $\tilde{\mathbf{p}}(t_{k+1})$, $\tilde{\mathbf{w}}(t_{k+1})$ on the new time step.

6. **Post-stabilization of generalized coordinates and velocities**. Using the post-stabilization described in the previous chapter, we obtain from $\tilde{\mathbf{p}}(t_{k+1})$, $\tilde{\mathbf{w}}(t_{k+1})$ the stabilized values of the generalized coordinates $\mathbf{p}(t_{k+1})$ and generalized velocities $\mathbf{w}(t_{k+1})$.

In this chapter we precisely observe the most important theoretical problems of our method for the distributed simulations of multibodies.

Remark 2.1 *We show our method in the case of conservative systems but it can be also extended for the simulation of non-conservative systems with various degrees of freedom.*

Remark 2.2 *For the sake of simplicity we assume that a ground can be included only on the highest level of the hierarchy. This limit can be easily removed through minor modifications of the method.*

In our implementation a ground object can be included on each hierarchy's level. In Chapter 7 we demonstrate the simulation of a multibody system where subsystems include ground objects.

2.2 Choice of coordinates

There are two main approaches for the generation of equations of motion: perform it using generalized coordinates or perform it using absolute coordinates. Both approaches have its advantages and disadvantages.

If we use generalized coordinates in the case of a loops-free model, then for many types of joints we do not need to stabilize a simulation model. In the case of a model with closed loops, use of generalized coordinates significantly reduces the post-stabilization complexity.

Example 2.1. *Consider a two-dimensional loop with* m *revolute joints. The dimension of* $\mathbf{G}$ *expressed using generalized coordinates is* [m, 2] *vs.* [3m, 2m] *of* $\mathbf{G}$ *expressed using absolute coordinates. This property is very important because in the post-stabilization we need to inverse the matrix* $\mathbf{G}\mathbf{G}^T$.

But if we use generalized coordinates in the calculation of accelerations and internal forces, then we cannot separate our system into subsystems. It happens because some of the generalized coordinates are included in the equations of motion of all bodies. Therefore, we need to perform our calculations of accelerations and internal forces using absolute coordinates.

Trying to maximise the effectiveness of the method, we use the combination of generalized and absolute coordinates. The distributive calculation of forces and accelerations we perform using absolute velocities and coordinates. But we perform the integration and stabilization steps using generalized accelerations. This combination leads some extra calculations needed for the transformation from absolute to generalized acceleration and from generalized to absolute coordinates. But this additional numerical complexity is much less than the numerical complexity of the post-stabilization using absolute coordinates. In Chapter 3 we precisely compare the effectiveness of this two stabilization's types.

2.3 Choice of absolute coordinates

Let us consider an arbitrary simulating body. Let k denote the number of the body. The vector $\mathbf{q}_k$ of absolute coordinates of the body consists of three Cartesian coordinates $\mathbf{x}_k = \begin{pmatrix} x_{k,1} & x_{k,2} & x_{k,3} \end{pmatrix}^T$ indicating the position of centre of mass of the body with respect to the global frame and a set of coordinates indicating the orientation of the body fixed frame with respect to the global frame. The orientation can be described by three angles (Eulerian angles) or by four Euler parameters [NIK 83, SHA 89, JAI 91, LUB 92].

In the case of using of Eulerian angles we obtain the significant computation difficulties when the mutation angle is equal to null. That is why in our method we use Euler parameters that do not have critical points. Using of Euler parameters is concerned with quaternions algebra discussed in Appendix A.

When four Euler parameters $\boldsymbol{\theta}_k = \begin{pmatrix} e_{k,0} & e_{k,1} & e_{k,2} & e_{k,3} \end{pmatrix}^T$ are used, a simple relationship exists between the components of the global angular velocity vector $\boldsymbol{\Omega}_k$ and time derivatives of Euler parameters $\dot{\boldsymbol{\theta}}_k = \begin{pmatrix} \dot{e}_{k,0} & \dot{e}_{k,1} & \dot{e}_{k,2} & \dot{e}_{k,3} \end{pmatrix}^T$:

$$\dot{\boldsymbol{\theta}}_k = \frac{1}{2}\mathbf{E}_k{}^T \boldsymbol{\Omega}_k$$

where $\mathbf{E}_k$ is a semi-transformation matrix [NIK 82] that depends linearly on Euler parameters:

$$\mathbf{E}_k = \begin{pmatrix} -e_{k,1} & e_{k,0} & -e_{k,3} & e_{k,2} \\ -e_{k,2} & e_{k,3} & e_{k,0} & -e_{k,1} \\ -e_{k,3} & -e_{k,2} & e_{k,1} & e_{k,0} \end{pmatrix}$$

The position variables are:

$$\mathbf{q}_k = \begin{pmatrix} \mathbf{x}_k{}^T & \boldsymbol{\theta}_k{}^T \end{pmatrix}^T$$

The velocity variables are:

$$\mathbf{v}_k = \begin{pmatrix} \dot{\mathbf{x}}_k{}^T & \boldsymbol{\Omega}_k{}^T \end{pmatrix}^T$$

The body position and velocity variables are related:

$$\dot{\mathbf{q}}_k = \begin{pmatrix} \dot{\mathbf{x}}_k \\ \dot{\boldsymbol{\theta}}_k \end{pmatrix} = \mathbf{T}_k \mathbf{v}_k = \begin{pmatrix} \mathbf{I}_3 & 0 \\ 0 & \frac{1}{2}\mathbf{E}_k^T \end{pmatrix} \mathbf{v}_k$$

where

$\mathbf{I}_3$ is the [3,3] identity matrix,

$\mathbf{T}_k$ is the [7,6] *velocity transformation matrix of the k-th body*.

Also exists the backward relation:

$$\mathbf{v}_k = \overline{\mathbf{T}}_k \cdot \dot{\mathbf{q}}_k = \begin{pmatrix} \mathbf{I_3} & \mathbf{0} \\ \mathbf{0} & \mathbf{2E}_k \end{pmatrix} \dot{\mathbf{q}}_k$$

where $\overline{\mathbf{T}}_k$ is the [6,7] *backward velocity transformation matrix of the k-th body*

Use of Euler parameters requires the normalization condition:

$$\left\| \boldsymbol{\theta}_k \right\| = 1$$

2.4 Calculation of absolute coordinates and velocities

From the object-oriented point of view the most convenient is to use generalized coordinates $\mathbf{p}$ and generalized velocities $\mathbf{w}$ associated with constraints.

Consider a constraint connecting a set of bodies $\boldsymbol{J}$={*Body* j_1, *Body* j_2,..., *Body* j_s}. If it exists a dependency of the coordinates $\mathbf{q}_\mathbf{K} = \left(\mathbf{q}_{k_1}{}^T \quad \mathbf{q}_{k_2}{}^T \quad \cdots \quad \mathbf{q}_{k_t}{}^T\right)^T$ of some subset of bodies $\boldsymbol{K}$={*Body* k_1, *Body* k_2,..., *Body* k_t} on the generalized coordinates $\mathbf{p}$ and on the coordinates $\mathbf{q}_\mathbf{B} = \left(\mathbf{q}_{b_1}{}^T \quad \mathbf{q}_{b_2}{}^T \quad \cdots \quad \mathbf{q}_{b_r}{}^T\right)^T$ of some other subset of bodies $\boldsymbol{B}$={*Body* b_1, *Body* b_2,..., *Body* b_r}:

$$\mathbf{q}_{\boldsymbol{K}} = \underline{q}(\mathbf{p}, \mathbf{q}_{\boldsymbol{B}})$$

then bodies *Body* b_1,..., *Body* b_r are called basic for the constraint, and bodies *Body* k_1, ..., *Body* k_t are called dependent on the constraint.

Example 2.2. *Consider a revolute joint connecting two bodies. Let* *Body 1* *be basic and* *Body 2* *be dependent. The generalized coordinate* p *associated with the joint is the angle between* *Body 1* *and* *Body 2. The generalized velocity* w *associated with the joint is the time derivative of* p.

The absolute coordinates of *Body 2* *are expressed as the function of the coordinates of the basic body and the angle between the connected bodies* p:

$$\mathbf{q}_2 = \underline{q}(p, \mathbf{q}_1) = \begin{pmatrix} \mathbf{x}_1 + \mathbf{A}_{0,1}(\boldsymbol{\theta}_1)\cdot\mathbf{r}_1 - \mathbf{A}_{0,1}(\boldsymbol{\theta}_1)\cdot\mathbf{A}_{1,2}(p)\cdot\mathbf{r}_2 \\ \boldsymbol{\theta}_1 \circ \mathbf{s}(p) \end{pmatrix}$$

where

$\mathbf{x}_1$ *are the coordinates of the centre of mass of* *Body 1,*

$\mathbf{r}_1$ *is the state vector expressed in the frame connected with* *Body 1* *from the centre of mass of the body to the centre of joint,*

$\mathbf{r}_2$ *is the state vector expressed in the frame connected with* *Body 2* *from the centre of mass of the body to the centre of joint,*

$\boldsymbol{\theta}_1$ *is the vector of Euler parameters of* $Body\ 1$,

$\mathbf{A}_{0,1}(\boldsymbol{\theta}_1)$ *is the matrix of rotation of* $Body\ 1$,

$\mathbf{A}_{1,2}(\mathrm{p})$ *is the matrix of relative rotation,*

$s = (s_0 \quad \mathbf{s}^T)^T = (\cos(p/2) \quad \sin(p/2)\mathbf{a}_1^T)^T$ *are Euler parameters describing the relative rotation around the axe* $\mathbf{a}_1$, *where* $\mathbf{a}_1$ *is the axe of relative rotation expressed in the frame connected with* $Body\ 1$.

Example 2.3. *Consider a ball joint connecting two bodies. Let* $Body\ 1$ *be basic and* $Body\ 2$ *be dependent. The set of generalized coordinates* $\mathbf{p} = (p_1 \quad p_2 \quad p_3 \quad p_4)^T$ *is equal to the vector of Euler parameters* $\boldsymbol{\theta}_2$ *of* $Body\ 2$. *The generalized velocity* $\mathbf{w}$ *is equal to the angular velocity* $\boldsymbol{\Omega}_1$ *of* $Body\ 1$.

For more details of descriptions of different types of joints, see Chapter 6.

Bodies that do not depend on any constraint are called *independent*. In Fig. 2.3 is shown the *graph of a multibody system.* The system consists of 7 bodies connected with two grounds. Grounds (i.e. bodies whose motion are predefined) are represented by red points, independent bodies are represented by blue points, and other bodies are represented by black points. Directed arcs stand for constraints that are used in transformations. Other constraints are represented by undirected arcs.

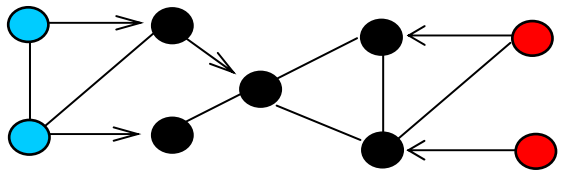

Figure 2.3: Graph of a 7-bodies system

While the translation of a multibody system we generate the *array of independent bodies* $\boldsymbol{I}=\{Body\ i_1, \ldots, Body\ i_n\}$ and the sequence of constraints $\boldsymbol{C}=\{Constraint\ c_1, .., Constraint\ c_m\}$ that are used during the transformation. Let us call $\boldsymbol{C}$ *the sequence of dependencies.* Obviously, $\mathbf{p}$ can be written as:

$$\mathbf{p} = \left(\mathbf{q_I}^T \quad \mathbf{p_C}^T\right)^T$$

where

$\mathbf{q}_\mathbf{I}$ is the vector of absolute coordinates of bodies in ***I,***

$\mathbf{p}_\mathbf{C}$ is the vector of generalized coordinates associated with constraints in ***C***.

On each time step we perform the same routine. At first we obtain the absolute coordinates of bodies included in ***I***. Then, using the ***C***-order, we consequently calculate the absolute coordinates of dependent bodies as the result of the $\underline{q}$-function of constraints. After the routine's completion we calculate the absolute coordinates of all simulating bodies.

Clearly, we have two limits on the structure of ***C***. The first is the limit on the set of constraints included in ***C***: a body can not be dependent on two different constraints. Else way we calculate two times the absolute coordinates of the body.

The second is the limit on the order of constraints inside ***C***. Consider an arbitrary *Body j*. Suppose that the body is dependent on *Constraint* c_1 and it is basic for *Constraint* c_2. If *Constraint* c_1 would be situated after *Constraint* c_2 in the sequence ***C,*** then we calculate the absolute coordinates of bodies that are dependent on *Constraint* c_2 before we calculate the coordinates of *Body j.* Therefore, *Constraint* c_1 should be before *Constraint* c_2.

Finally, we obtain that ***C*** has a tree-structure without loop-closing constraints.

We perform the calculation of absolute velocities in a similar way as the calculation of the absolute coordinates. For the calculations of velocities of bodies dependent on a constraint we use the constraint's function $\underline{v}$ equal to the time derivative of $\underline{q}$:

$$\underline{v}(\mathbf{p}, \mathbf{q}_\mathbf{B}, \mathbf{w}, \mathbf{v}_\mathbf{B}) = \dot{\underline{q}}$$

2.5 Equations of motion of a basic subsystem

Consider a basic subsystem S, shown in Fig. 2.4, included in a complete simulating system. By n denote the number of bodies in S. Let $\mathbf{g}$ denote the vector of equations of internal constraints:

$$\mathbf{g} = \begin{pmatrix} g_1(\mathbf{q}) & \dots & g_c(\mathbf{q}) \end{pmatrix}^T = \begin{pmatrix} 0 & \dots & 0 \end{pmatrix}^T \tag{2.1}$$

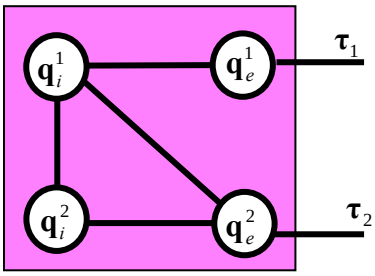

Figure 2.4: A subsystem of several connected bodies

Let $\boldsymbol{\tau}$ be the vector of Lagrange forces acting in external constraints. Then the descriptor form of equations of motion can be written as [CHI 95], [STE 01]:

$$\begin{aligned} &\dot{\mathbf{q}} = \mathbf{T}(\mathbf{q})\mathbf{v} \\ &\mathbf{M}(\mathbf{q})\dot{\mathbf{v}} + \mathbf{c}(\mathbf{q}) = \mathbf{G}(\mathbf{q})^T\boldsymbol{\lambda} + \boldsymbol{\tau} \\ &\mathbf{g}(\mathbf{q}) = 0 \\ &\left\|\boldsymbol{\theta}_k\right\| = 1 \qquad k = 1...n \end{aligned} \tag{2.2}$$

where

$$\mathbf{T} = diag(\mathbf{T}_1,...,\mathbf{T}_n) \qquad \mathbf{T}_k = \begin{pmatrix} \mathbf{I}_3 & 0 \\ 0 & \frac{1}{2}\mathbf{E}_k^T \end{pmatrix}$$

$$\mathbf{M} = diag(\mathbf{M}_1,...,\mathbf{M}_n) \qquad \mathbf{M}_k = \begin{pmatrix} m_k\mathbf{I}_3 & 0 \\ 0 & \mathbf{J}_k \end{pmatrix}$$

$$\mathbf{c} = \begin{pmatrix} \mathbf{c}_1^T & \cdots & \mathbf{c}_n^T \end{pmatrix}^T \qquad \mathbf{c}_k = \begin{pmatrix} -\mathbf{f}_k \\ -\mathbf{l}_k + \boldsymbol{\Omega}_k \times \mathbf{J}_k \cdot \boldsymbol{\Omega}_k \end{pmatrix}$$

$$\mathbf{G} = \left(\frac{\partial \mathbf{g}}{\partial \mathbf{q}}\right)\mathbf{T}$$

Here

$\mathbf{f}_k$ is the resultant external force acting on the k-th body,

$\mathbf{l}_k$ is the resultant external torque acting on the k-th body,

m_k is the mass of the k-th body,

$\mathbf{M}$ is the mass matrix,

$\mathbf{J}_k$ is the [3,3] moment of inertia matrix of the k-th body with respect to the *body centre-of-mass frame*,

$\mathbf{I}_3$ is the [3,3] identity matrix.

Here a *body centre-of-mass frame* is a frame parallel to inertial frame but centred at body centre of mass.

Remark 2.3 *Matrices* $\mathbf{J}_k$ *are not constant and should be calculated on each time step from formula* [WIT 77]*:*

$$\mathbf{J}_k = \mathbf{A}_{0,k}(\mathbf{q}_k) \cdot \bar{\mathbf{J}}_k \cdot \mathbf{A}_{k,0}(\mathbf{q}_k)$$

where

$\mathbf{A}_{0,k}(\mathbf{q}_k)$ *is the rotation matrix of the* k*-th body,*

$\mathbf{A}_{k,0}(\mathbf{q}_k) = \mathbf{A}_{0,k}^T(\mathbf{q}_k)$ *is the backward rotation matrix of the* k*-th body,*

$\bar{\mathbf{J}}_k$ *is the constant* [3,3] *moment of inertia matrix of the* k*-th body expressed in the body-fixed frame centred at body centre of mass.*

Let first m bodies are connected with the complete system by external joints. Let $\mathbf{q}_e$ denote the $7m$-length vector of absolute coordinates of bordering bodies. Let $\mathbf{q}_i$ denote the $7(n\text{-}m)$-length vector of absolute coordinates of internal bodies. Obviously, $\mathbf{q}$ can be written as:

$$\mathbf{q} = \begin{pmatrix} \mathbf{q_e}^T & \mathbf{q_i}^T \end{pmatrix}^T \tag{2.3}$$

Therefore, we can write (2.2) in the new form:

$$\dot{\mathbf{q}}_\mathbf{i} = \mathbf{T}_\mathbf{i}\mathbf{v}_\mathbf{i} \tag{2.4}$$

$$\dot{\mathbf{q}}_\mathbf{e} = \mathbf{T}_\mathbf{e}\mathbf{v}_\mathbf{e} \tag{2.5}$$

$$\mathbf{M}_\mathbf{e}\dot{\mathbf{v}}_\mathbf{e} + \mathbf{c}_\mathbf{e} = \mathbf{G}_\mathbf{e}^{\mathrm{T}}\boldsymbol{\lambda} + \boldsymbol{\tau} \tag{2.6}$$

$$\mathbf{M}_\mathbf{i}\dot{\mathbf{v}}_\mathbf{i} + \mathbf{c}_\mathbf{i} = \mathbf{G}_\mathbf{i}^{\mathrm{T}}\boldsymbol{\lambda} \tag{2.7}$$

$$\mathbf{g}(\mathbf{q}_\mathbf{e}, \mathbf{q}_\mathbf{i}) = \mathbf{0} \tag{2.8}$$

$$\|\boldsymbol{\theta}_k\| = \mathbf{1} \qquad k = 1...n \tag{2.9}$$

where

$$\begin{aligned}
\mathbf{T}_\mathbf{e} &= diag(\mathbf{T}_1,...,\mathbf{T}_m)\\
\mathbf{T}_\mathbf{i} &= diag(\mathbf{T}_{m+1},...,\mathbf{T}_n)\\
\mathbf{M}_\mathbf{e} &= diag(\mathbf{M}_1,...,\mathbf{M}_m)\\
\mathbf{M}_\mathbf{i} &= diag(\mathbf{M}_{m+1},...,\mathbf{M}_n)\\
\mathbf{G}_\mathbf{e} &= \left(\frac{\partial \mathbf{g}}{\partial \mathbf{q}_\mathbf{e}}\right)\mathbf{T}_\mathbf{e} \qquad \mathbf{G}_\mathbf{i} = \left(\frac{\partial \mathbf{g}}{\partial \mathbf{q}_\mathbf{i}}\right)\mathbf{T}_\mathbf{i}
\end{aligned}$$

Thus, differentiating (2.1) once, we obtain the constraint equations on velocity level:

$$\mathbf{0} = \frac{\partial \mathbf{g}}{\partial \mathbf{q}}\dot{\mathbf{q}} = \frac{\partial \mathbf{g}}{\partial \mathbf{q}}\mathbf{T}\mathbf{v} = \mathbf{G}\mathbf{v} \tag{2.10}$$

and further differentiation with respect to time results in the constraint equations on acceleration level:

$$\mathbf{0} = \mathbf{G}\dot{\mathbf{v}} + \dot{\mathbf{G}}\mathbf{v} = \begin{pmatrix}\mathbf{G}_\mathbf{e} & \mathbf{G}_\mathbf{i}\end{pmatrix}\begin{pmatrix}\dot{\mathbf{v}}_\mathbf{e}\\ \dot{\mathbf{v}}_\mathbf{i}\end{pmatrix} + \mathbf{u} = \mathbf{G}_\mathbf{e}\dot{\mathbf{v}}_\mathbf{e} + \mathbf{G}_\mathbf{i}\dot{\mathbf{v}}_\mathbf{i} + \mathbf{u} \tag{2.11}$$

Substituting $\dot{\mathbf{v}}_\mathbf{e}$, $\dot{\mathbf{v}}_\mathbf{i}$ from (2.6), (2.7), we obtain:

$$\mathbf{0} = \mathbf{G}_\mathbf{e}\mathbf{M}_\mathbf{e}^{-1}(\mathbf{G}_\mathbf{e}^{\mathrm{T}}\boldsymbol{\lambda} + \boldsymbol{\tau} - \mathbf{c}_\mathbf{e}) + \mathbf{G}_\mathbf{i}\mathbf{M}_\mathbf{i}^{-1}(\mathbf{G}_\mathbf{i}^{\mathrm{T}}\boldsymbol{\lambda} - \mathbf{c}_\mathbf{i}) + \mathbf{u}$$

that can be rewritten as

$$\mathbf{0} = \mathbf{G}\mathbf{M}^{-1}\mathbf{G}^{\mathrm{T}}\boldsymbol{\lambda} + \mathbf{G}_\mathbf{e}\mathbf{M}_\mathbf{e}^{-1}\boldsymbol{\tau} - \mathbf{G}\mathbf{M}^{-1}\mathbf{c} + \mathbf{u} \tag{2.12}$$

yielding the dependency of Lagrange multipliers $\boldsymbol{\lambda}$ on forces $\boldsymbol{\tau}$ in external links:

$$\boldsymbol{\lambda} = \mathbf{S}\boldsymbol{\tau} + \mathbf{b} \tag{2.13}$$

where

$$\begin{aligned}
\mathbf{S} &= -(\mathbf{G}\mathbf{M}^{-1}\mathbf{G}^{\mathrm{T}})^{-1}\mathbf{G}_\mathbf{e}\mathbf{M}_\mathbf{e}^{-1}\\
\mathbf{b} &= (\mathbf{G}\mathbf{M}^{-1}\mathbf{G}^{\mathrm{T}})^{-1}(\mathbf{G}\mathbf{M}^{-1}\mathbf{c} - \mathbf{u})
\end{aligned} \tag{2.14}$$

If $\mathbf{G}$ does not have dependent rows, then we can invert $(\mathbf{G}\mathbf{M}^{-1}\mathbf{G}^{\mathrm{T}})$ because $\mathbf{M}$ is positive definite.

Substituting $\boldsymbol{\lambda}$ in (2.6), we obtain the relation between $\dot{\mathbf{v}}_\mathbf{e}$ and $\boldsymbol{\tau}$:

$$\dot{\mathbf{v}}_e = \mathbf{D}\boldsymbol{\tau} + \mathbf{r} \tag{2.15}$$

where

$$\begin{aligned} \mathbf{D} &= \mathbf{M}_e^{-1}\mathbf{G}_e^{T}\mathbf{S} + \mathbf{M}_e^{-1} \\ \mathbf{r} &= \mathbf{M}_e^{-1}\mathbf{G}_e^{T}\mathbf{b} - \mathbf{M}_e^{-1}\mathbf{c}_e \end{aligned} \tag{2.16}$$

If we know $\boldsymbol{\tau}$, then we can also calculate the accelerations of internal bodies: using equation (2.13), we get the value of Lagrange multipliers $\boldsymbol{\lambda}$ and then substitute it to the modification of equation (2.4):

$$\dot{\mathbf{v}}_i = \mathbf{M}_i^{-1}(\mathbf{G}_i^{T}\boldsymbol{\lambda} - \mathbf{c}_i) \tag{2.17}$$

We use this property after we obtain $\boldsymbol{\tau}$.

Remark 2.4 *Consider the case when* $\mathbf{G}$ *has dependent rows. Let* $\tilde{\mathbf{G}}$ *denote the matrix obtained from* $\mathbf{G}$ *by elimination of dependent rows. Obviously, we can represent* $\tilde{\mathbf{G}}$ *as*:

$$\mathbf{G} = \mathbf{R}\tilde{\mathbf{G}}$$

where $\mathbf{R}$ *is the dependency matrix.*

Then (2.6), (2.7) *can be rewritten:*

$$\begin{aligned} \mathbf{M}_e\dot{\mathbf{v}}_e + \mathbf{c}_e &= \tilde{\mathbf{G}}_e^{T}\boldsymbol{\mu} + \boldsymbol{\tau} \\ \mathbf{M}_i\dot{\mathbf{v}}_i + \mathbf{c}_i &= \tilde{\mathbf{G}}_i^{T}\boldsymbol{\mu} \end{aligned} \tag{2.18}$$

where $\boldsymbol{\mu}$ *is the vector of new Lagrange multipliers:*

$$\boldsymbol{\mu} = \mathbf{R}^{T}\boldsymbol{\lambda}$$

Now we can obtain $\mathbf{D}$ *and* $\mathbf{r}$ *in the same way as it was described above. We need only to substitute in equation* (2.11) - (2.17) $\tilde{\mathbf{G}}$ *instead of* $\mathbf{G}$ *and* $\boldsymbol{\mu}$ *instead of* $\boldsymbol{\lambda}$.

2.6 Building up the hierarchy

Consider a derived subsystem S consisting of N parent subsystems: $S_1, S_2, \ldots, S_N$, shown in Fig. 2.5. Let $\mathbf{q}_E$ denote the vector of coordinates of bodies bordered to the

parents of S. Since the definition of bordering bodies, it follows that the vector $\mathbf{q}_E$ is the union of vectors $\mathbf{q}_e^{(k)}$ (k=1..N).

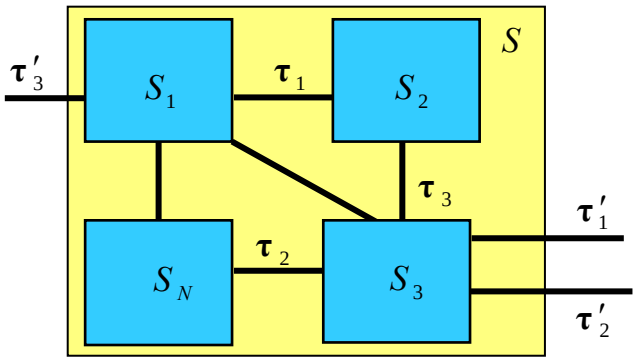

Figure 2.5: A subsystem consisting of several connected subsystems

Let $\mathbf{q}_{Ext} \subset \mathbf{q}_E$ be the vector of coordinates of bodies bordered to S. Let $\mathbf{q}_{Ein} \subset \mathbf{q}_E$ denote the vector of coordinates of bodies internal to S. Obviously, $\mathbf{q}_E$ can be written as:

$$\mathbf{q}_{\mathbf{E}} = (\mathbf{q}_{\mathbf{Ext}}^T \quad \mathbf{q}_{\mathbf{Ein}}^T)^T$$

Let $\mathbf{g}$ denote the vector of equations of internal constraints between $S_1, S_2, \ldots, S_N$:

$$\mathbf{g} = \begin{pmatrix} g_1(\mathbf{q}_{\mathbf{Ext}}, \mathbf{q}_{\mathbf{Ein}}) & \cdots & g_c(\mathbf{q}_{\mathbf{Ext}}, \mathbf{q}_{\mathbf{Ein}}) \end{pmatrix}^T = \begin{pmatrix} 0 & \ldots & 0 \end{pmatrix}^T \tag{2.19}$$

By $\mathbf{G}$ denote the constraint Jacobian matrix multiplied by the matrix $\mathbf{T}$:

$$\mathbf{G} = (\mathbf{G}_{\mathbf{Ext}} \quad \mathbf{G}_{\mathbf{Ein}}) = \left(\frac{\partial \mathbf{g}}{\partial \mathbf{q}_{\mathbf{Ext}}} \quad \frac{\partial \mathbf{g}}{\partial \mathbf{q}_{\mathbf{Ein}}} \right) \mathbf{T}$$

Let $\boldsymbol{\lambda}$ denote the vector of Lagrange multipliers associated with the constraints between subsystems $S_1, S_2, \ldots, S_N$. Denote the forces acting in links external to S as $\boldsymbol{\tau}'$. From the previous hierarchy level we get matrices $\mathbf{D}^{(k)}$ and vectors $\mathbf{r}^{(k)}$. We can unite the equations of accelerations

$$\dot{\mathbf{v}}_{\mathbf{E}}^{(k)} = \mathbf{D}^{(k)} \boldsymbol{\tau}^{(k)} + \mathbf{r}^{(k)} \qquad k = 1 \ldots N \tag{2.20}$$

in two matrix equations:

$$\begin{aligned} \dot{\mathbf{v}}_{\mathbf{Ext}} &= \hat{\mathbf{D}}_{\mathbf{Ext}} \mathbf{G}^T \boldsymbol{\lambda} + \hat{\mathbf{D}}'_{\mathbf{Ext}} \boldsymbol{\tau}' + \hat{\mathbf{r}}_{\mathbf{Ext}} \\ \dot{\mathbf{v}}_{\mathbf{Ein}} &= \hat{\mathbf{D}}_{\mathbf{Ein}} \mathbf{G}^T \boldsymbol{\lambda} + \hat{\mathbf{D}}'_{\mathbf{Ein}} \boldsymbol{\tau}' + \hat{\mathbf{r}}_{\mathbf{Ein}} \end{aligned} \tag{2.21}$$

or, in the other form:

$$\dot{\mathbf{v}}_{\mathbf{e}} = \hat{\mathbf{D}}\mathbf{G}^{\mathbf{T}}\boldsymbol{\lambda} + \hat{\mathbf{D}}'\boldsymbol{\tau}' + \hat{\mathbf{r}} \tag{2.22}$$

While obtaining $\boldsymbol{\lambda}$ we need to invert the matrix $\mathbf{G}\hat{\mathbf{D}}\mathbf{G}^{T}$. In section 2.5 it was demonstrated that we can eliminate dependent rows from $\mathbf{G}$, but the problem is that $\hat{\mathbf{D}}$ can be singular. That is why we use the reduction of eigendecomposition of $\hat{\mathbf{D}}$.

Let r denote the size of $\hat{\mathbf{D}}$. Then we can rewrite $\hat{\mathbf{D}}$ in the form

$$\hat{\mathbf{D}} = \mathbf{Z}\tilde{\mathbf{D}}\mathbf{Z}^{\mathbf{T}} \tag{2.23}$$

where $\tilde{\mathbf{D}}$ is a diagonal matrix composed of m nonzero eigenvalues of $\hat{\mathbf{D}}$:

$$\tilde{\mathbf{D}} = \begin{pmatrix} \zeta_1 & 0 & \cdots & 0 \\ 0 & \zeta_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \zeta_m \end{pmatrix}$$

and $\mathbf{Z}$ is a matrix composed of eigenvectors:

$$\mathbf{Z} = \begin{pmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \dots & \mathbf{z}_m \end{pmatrix} = \begin{pmatrix} z_{1,1} & z_{1,2} & \cdots & z_{1,m} \\ z_{2,1} & z_{2,2} & \cdots & z_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ z_{r,1} & z_{r,2} & \cdots & z_{r,m} \end{pmatrix}$$

All eigenvalues of $\hat{\mathbf{D}}$ are real because $\hat{\mathbf{D}}$ is a symmetric matrix.

Let $\tilde{\mathbf{G}}$ denote the matrix obtained from the matrix $\mathbf{GZ}$ by elimination of dependent rows. Obviously, we can represent $\tilde{\mathbf{G}}$ as:

$$\begin{aligned} \tilde{\mathbf{G}} &= \mathbf{K}\mathbf{G}\mathbf{Z} \\ \mathbf{G}\mathbf{Z} &= \mathbf{N}\tilde{\mathbf{G}} \end{aligned} \tag{2.24}$$

where $\mathbf{N}$ and $\mathbf{K}$ are dependency matrices.

Now we can rewrite equation (2.22) in the form:

$$\dot{\mathbf{v}}_{\mathbf{E}} = \mathbf{Z}\tilde{\mathbf{D}}\tilde{\mathbf{G}}^{\mathbf{T}}\boldsymbol{\mu} + \hat{\mathbf{D}}'\boldsymbol{\tau}' + \hat{\mathbf{r}} \tag{2.25}$$

or, separately:

$$\dot{\mathbf{v}}_{\mathbf{Ext}} = \mathbf{Z}_{\mathbf{Ext}}\tilde{\mathbf{D}}\tilde{\mathbf{G}}^{\mathrm{T}}\boldsymbol{\mu} + \hat{\mathbf{D}}'_{\mathbf{Ext}}\boldsymbol{\tau}' + \hat{\mathbf{r}}_{\mathbf{Ext}} \tag{2.26}$$

$$\dot{\mathbf{v}}_{\mathbf{Ein}} = \mathbf{Z}_{\mathbf{Ein}}\tilde{\mathbf{D}}\tilde{\mathbf{G}}^{\mathrm{T}}\boldsymbol{\mu} + \hat{\mathbf{D}}'_{\mathbf{Ein}}\boldsymbol{\tau}' + \hat{\mathbf{r}}_{\mathbf{Ein}} \tag{2.27}$$

where $\boldsymbol{\mu}$ are new Lagrange multipliers:

$$\boldsymbol{\mu} = \mathbf{N}^{T}\boldsymbol{\lambda}$$

Differentiating (2.19) once, we obtain the constraint equations on velocity level:

$$\mathbf{0} = \frac{\partial \mathbf{g}}{\partial \mathbf{q}_{\mathbf{E}}}\dot{\mathbf{q}}_{\mathbf{E}} = \frac{\partial \mathbf{g}}{\partial \mathbf{q}_{\mathbf{E}}}\mathbf{T}\mathbf{v}_{\mathbf{E}} = \mathbf{G}\mathbf{v}_{\mathbf{E}}$$

and further differentiation with respect to time and multiplication by $\mathbf{K}$ results in the constraint equations on acceleration level:

$$\mathbf{0} = \mathbf{K}\mathbf{G}\dot{\mathbf{v}}_{\mathbf{E}} + \mathbf{K}\dot{\mathbf{G}}\mathbf{v}_{\mathbf{E}} = \mathbf{K}\mathbf{G}\dot{\mathbf{v}}_{\mathbf{E}} + \mathbf{u}$$

Substituting (2.25), we get:

$$\mathbf{0} = \mathbf{K}\mathbf{G}(\mathbf{Z}\tilde{\mathbf{D}}\tilde{\mathbf{G}}^{\mathrm{T}}\boldsymbol{\mu} + \hat{\mathbf{D}}'\boldsymbol{\tau}' + \hat{\mathbf{r}}) + \mathbf{u}$$

Now we obtain the dependency of Lagrange multipliers $\boldsymbol{\mu}$ on $\boldsymbol{\tau}'$:

$$\boldsymbol{\mu} = \mathbf{S}\boldsymbol{\tau}' + \mathbf{b} \tag{2.28}$$

where

$$\begin{aligned} \mathbf{S} &= -(\tilde{\mathbf{G}}\tilde{\mathbf{D}}\tilde{\mathbf{G}}^{\mathrm{T}})^{-1}\mathbf{K}\mathbf{G}\hat{\mathbf{D}}' \\ \mathbf{b} &= -(\tilde{\mathbf{G}}\tilde{\mathbf{D}}\tilde{\mathbf{G}}^{\mathrm{T}})^{-1}(\mathbf{K}\mathbf{G}\hat{\mathbf{r}} + \mathbf{u}) \end{aligned} \tag{2.29}$$

Substituting $\boldsymbol{\mu}$ in (2.26), we obtain:

$$\dot{\mathbf{v}}_{\mathbf{Ext}} = \mathbf{D}\boldsymbol{\tau}' + \mathbf{r} \tag{2.30}$$

where

$$\begin{aligned} \mathbf{D} &= \mathbf{Z}_{\mathbf{Ext}}\tilde{\mathbf{D}}\tilde{\mathbf{G}}^{\mathrm{T}}\mathbf{S} + \hat{\mathbf{D}}'_{\mathbf{Ext}} \\ \mathbf{r} &= \mathbf{Z}_{\mathbf{Ext}}\tilde{\mathbf{D}}\tilde{\mathbf{G}}^{\mathrm{T}}\mathbf{b} + \hat{\mathbf{r}}_{\mathbf{Ext}} \end{aligned} \tag{2.31}$$

If we know $\boldsymbol{\tau}'$, then we can also calculate the accelerations $\dot{\mathbf{v}}_{\mathbf{Ein}}$ of internal bodies: using equation (2.28), we get the values of Lagrange multipliers $\boldsymbol{\mu}$ and then substitute it in equation (2.27). We use this property after we obtain the forces $\boldsymbol{\tau}'$.

We should iteratively perform this step of the simulation for the next levels of the hierarchy until the subsystem includes all bodies.

2.7 Calculation of absolute accelerations

Consider a system S of the highest hierarchy level. Suppose that the system consists of N parent subsystems $S_1, S_2, \ldots, S_N$ and the ground whose absolute coordinates $\mathbf{q}_0$ are predefined: $\mathbf{q}_0 = \mathbf{q}_0(t)$.

Remark 2.5 *The situation when the ground is not included in the complete system can also be easily described with the minor modifications of the formulas.*

Let $\mathbf{q}_{Ein}$ denote the vector of coordinates of bodies bordered to the parents of S ($\mathbf{q}_{Ein}$ is equal to $\mathbf{q}_E$ because S does not have external constraints). Obviously, the vector $\mathbf{q}_E$ is the union of vectors $\mathbf{q}_E^{(k)}$ ($k = 1 \ldots N$).

Let $\mathbf{g}$ denote the vector of equations of internal constraints:

$$\mathbf{g} = \begin{pmatrix} g_1(\mathbf{q}_0, \mathbf{q}_{\mathbf{Ein}}) & \ldots & g_c(\mathbf{q}_0, \mathbf{q}_{\mathbf{Ein}}) \end{pmatrix}^T = \begin{pmatrix} 0 & \ldots & 0 \end{pmatrix}^T \tag{2.32}$$

By $\mathbf{G}$ denote the constraint Jacobian matrix multiplied by the matrix $\mathbf{T}$:

$$\mathbf{G} = \begin{pmatrix} \mathbf{G}_0 & \mathbf{G}_{\mathbf{Ein}} \end{pmatrix} = \begin{pmatrix} \dfrac{\partial \mathbf{g}}{\partial \mathbf{q}_0} & \dfrac{\partial \mathbf{g}}{\partial \mathbf{q}_{\mathbf{Ein}}} \end{pmatrix} \mathbf{T}$$

Let $\boldsymbol{\lambda}$ denote the Lagrange multipliers associated with the constraints. From the previous hierarchy level we get matrices $\mathbf{D}^{(k)}$ and $\mathbf{r}^{(k)}$. We can unite the equations of accelerations

$$\begin{aligned} &\dot{\mathbf{v}}_{\mathbf{e}}^{(k)} = \mathbf{D}^{(k)} \boldsymbol{\tau}^{(k)} + \mathbf{r}^{(k)} \qquad k = 1 \ldots N \\ &\dot{\mathbf{v}}_0 = \dot{\mathbf{v}}_0(t) \end{aligned} \tag{2.33}$$

in two matrix equations:

$$\begin{aligned}\dot{\mathbf{v}}_{\mathbf{Ein}} &= \hat{\mathbf{D}}\mathbf{G}_{\mathbf{Ein}}^{T}\boldsymbol{\lambda} + \hat{\mathbf{r}} \\ \dot{\mathbf{v}}_{0} &= \dot{\mathbf{v}}_{0}(t)\end{aligned} \tag{2.34}$$

While obtaining $\boldsymbol{\lambda}$ we need to invert the matrix $\mathbf{G}_{\mathbf{Ein}}\hat{\mathbf{D}}\mathbf{G}_{\mathbf{Ein}}^{T}$. If $\hat{\mathbf{D}}$ is singular, then we use the reduction of the eigendecomposition of $\hat{\mathbf{D}}$ in the same way as it was performed while building up the hierarchy:

$$\hat{\mathbf{D}} = \mathbf{Z}\tilde{\mathbf{D}}\mathbf{Z}^{\mathbf{T}}$$

where $\tilde{\mathbf{D}}$ is a diagonal matrix composed of nonzero eigenvalues of $\hat{\mathbf{D}}$ and $\mathbf{Z}$ is a matrix composed of eigenvectors.

Let $\tilde{\mathbf{G}}_{\mathbf{Ein}}$ denote the matrix obtained from the matrix $\mathbf{G}_{\mathbf{Ein}}\mathbf{Z}$ by elimination of dependent rows. Obviously, we can represent $\tilde{\mathbf{G}}_{\mathbf{Ein}}$ as:

$$\begin{aligned}\tilde{\mathbf{G}}_{\mathbf{Ein}} &= \mathbf{K}\mathbf{G}_{\mathbf{Ein}}\mathbf{Z} \\ \mathbf{G}_{\mathbf{Ein}}\mathbf{Z} &= \mathbf{N}\tilde{\mathbf{G}}_{\mathbf{Ein}}\end{aligned}$$

where $\mathbf{N}$ and $\mathbf{K}$ are dependency matrices. Obviously,

$$\mathbf{N}\mathbf{K} = \mathbf{I} \tag{2.35}$$

where $\mathbf{I}$ is the identity matrix.

Now we can rewrite equation (2.34) in the form

$$\begin{aligned}\dot{\mathbf{v}}_{\mathbf{Ein}} &= \mathbf{Z}\tilde{\mathbf{D}}\tilde{\mathbf{G}}_{\mathbf{Ein}}^{T}\boldsymbol{\mu} + \hat{\mathbf{r}} \\ \dot{\mathbf{v}}_{0} &= \dot{\mathbf{v}}_{0}(t)\end{aligned} \tag{2.36}$$

where $\boldsymbol{\mu}$ are new Lagrange multipliers:

$$\boldsymbol{\mu} = \mathbf{N}^{T}\boldsymbol{\lambda} \tag{2.37}$$

Differentiating (2.32), we get:

$$\mathbf{0} = \begin{pmatrix} \dfrac{\partial \mathbf{g}}{\partial \mathbf{q}_0} & \dfrac{\partial \mathbf{g}}{\partial \mathbf{q}_{\mathbf{Ein}}} \end{pmatrix} \begin{pmatrix} \dot{\mathbf{q}}_0 \\ \dot{\mathbf{q}}_{\mathbf{Ein}} \end{pmatrix} = \begin{pmatrix} \dfrac{\partial \mathbf{g}}{\partial \mathbf{q}_0} & \dfrac{\partial \mathbf{g}}{\partial \mathbf{q}_{\mathbf{Ein}}} \end{pmatrix} \begin{pmatrix} \mathbf{T}_0\mathbf{v}_0 \\ \mathbf{T}_{\mathbf{Ein}}\mathbf{v}_{\mathbf{Ein}} \end{pmatrix} = \begin{pmatrix} \mathbf{G}_0 & \mathbf{G}_{\mathbf{Ein}} \end{pmatrix} \begin{pmatrix} \mathbf{v}_0 \\ \mathbf{v}_{\mathbf{Ein}} \end{pmatrix}$$

Differentiating this equation and multiplying by matrix $\mathbf{K}$, we obtain:

$$\mathbf{0} = \mathbf{K}\left[\begin{pmatrix} \dot{\mathbf{G}}_0 & \dot{\mathbf{G}}_{\mathbf{Ein}} \end{pmatrix} \begin{pmatrix} \mathbf{v}_0 \\ \mathbf{v}_{\mathbf{Ein}} \end{pmatrix} + \begin{pmatrix} \mathbf{G}_0 & \mathbf{G}_{\mathbf{Ein}} \end{pmatrix} \begin{pmatrix} \dot{\mathbf{v}}_0 \\ \dot{\mathbf{v}}_{\mathbf{Ein}} \end{pmatrix} \right] = \mathbf{K}\mathbf{G}_0\dot{\mathbf{v}}_0 + \mathbf{K}\mathbf{G}_{\mathbf{Ein}}\dot{\mathbf{v}}_{\mathbf{Ein}} + \mathbf{u}$$

Substituting $\dot{\mathbf{v}}_{\mathbf{Ein}}$ from (2.36), we get:

$$\mathbf{0} = \mathbf{K}\mathbf{G}_0\dot{\mathbf{v}}_0 + \mathbf{K}\mathbf{G}_{\mathbf{Ein}}\left(\mathbf{Z}\tilde{\mathbf{D}}\tilde{\mathbf{G}}_{\mathbf{Ein}}^T\boldsymbol{\mu} + \hat{\mathbf{r}}\right) + \mathbf{u} = \mathbf{K}\mathbf{G}_0\dot{\mathbf{v}}_0 + \tilde{\mathbf{G}}_{\mathbf{Ein}}\tilde{\mathbf{D}}\tilde{\mathbf{G}}_{\mathbf{Ein}}^T\boldsymbol{\mu} + \mathbf{K}\mathbf{G}_{\mathbf{Ein}}\hat{\mathbf{r}} + \mathbf{u}$$

Finally, we calculate $\boldsymbol{\mu}$:

$$\boldsymbol{\mu} = -\left(\tilde{\mathbf{G}}_{\mathbf{Ein}}\tilde{\mathbf{D}}\tilde{\mathbf{G}}_{\mathbf{Ein}}^T\right)^{-1}\left(\mathbf{K}\mathbf{G}_0\dot{\mathbf{v}}_0 + \mathbf{K}\mathbf{G}_{\mathbf{Ein}}\hat{\mathbf{r}} + \mathbf{u}\right) \tag{2.38}$$

From (2.35), (2.37) follows that

$$\boldsymbol{\lambda} = \mathbf{K}^T\boldsymbol{\mu}$$

Then we obtain the values of forces $\boldsymbol{\tau}$ acting in system's constraints:

$$\boldsymbol{\tau} = \mathbf{G}^T\boldsymbol{\lambda} = \mathbf{G}^T\mathbf{K}^T\boldsymbol{\mu}$$

and transmit them to the parents $S_1, S_2, \ldots S_N$.

In the previous step we obtained the relation between the accelerations of internal bodies and the forces in external links. Iteratively substituting the forces in external links to the previous levels of the hierarchy we obtain the absolute accelerations of all bodies.

2.8 Calculation of generalized accelerations

We perform the calculation of generalized accelerations similar to the calculation of absolute coordinates. For the calculation of generalized accelerations $\dot{\mathbf{w}}$ associated with a constraint we use the constraint's function $\underline{\varepsilon}$:

$$\underline{\varepsilon}(\mathbf{q}, \mathbf{v}, \dot{\mathbf{v}}) = \dot{\mathbf{w}}$$

where

$\mathbf{q}$ is the vector of absolute coordinates of connected bodies,

$\mathbf{v}$ is the vector of absolute velocities of connected bodies,

$\dot{\mathbf{v}}$ is the vector of absolute accelerations of connected bodies.

On each time step we obtain at first the generalized accelerations of objects included in $\boldsymbol{I}$ and then, using $\boldsymbol{C}$-order, we calculate the generalized accelerations associated with constraints from the constraint's function $\underline{\varepsilon}$. After finishing the routine we calculate all generalized accelerations.

2.9 Post-stabilization of generalized coordinates and velocities

After calculating the generalized accelerations $\dot{\mathbf{w}}$ we calculate the values of the generalized coordinates $\tilde{\mathbf{p}}(t_{k+1})$ and velocities $\tilde{\mathbf{w}}(t_{k+1})$ on the next time step using an ODE integration scheme (e.g. Runge-Kutta or multistep).

If our simulated system does not have closed loops, then usually we do not need to perform the stabilization because we use the generalized coordinates in the integration. Otherwise we should stabilize our solution trying to minimise the drift of the system. In our method we use the post-stabilization described in Chapter 1.

Let $\mathbf{A}^+$ denote the pseudoinverse of a matrix $\mathbf{A}$:

$$\mathbf{A}^+ = \mathbf{A}^T \left(\mathbf{A}\mathbf{A}^T\right)^{-1}$$

The stabilization equations [AHR 95] can be rewritten as:

$$\begin{aligned} \begin{pmatrix} \mathbf{p} \\ \mathbf{w} \end{pmatrix} &= \begin{pmatrix} \tilde{\mathbf{p}} \\ \tilde{\mathbf{w}} \end{pmatrix} - \begin{pmatrix} \Delta\mathbf{p} \\ \Delta\mathbf{w} \end{pmatrix} \\ \Delta\mathbf{p} &= \left(\frac{\partial \mathbf{g}(\tilde{\mathbf{p}})}{\partial \tilde{\mathbf{p}}} \right)^+ \mathbf{g}(\tilde{\mathbf{p}}) \\ \Delta\mathbf{w} &= \left(\mathbf{G}(\tilde{\mathbf{p}})\right)^+ \mathbf{G}(\tilde{\mathbf{p}})\tilde{\mathbf{w}} \end{aligned} \tag{2.39}$$

where

$\mathbf{g}(\tilde{\mathbf{p}})$ is the drift of non-trivial position constraints,

$\mathbf{G}(\tilde{\mathbf{p}})\tilde{\mathbf{w}}$ is the drift of non-trivial velocity constraints,

$\Delta\mathbf{p}$ is the vector stabilizing position constraints,

$\Delta\mathbf{w}$ is the vector stabilizing velocity constraints $\mathbf{G}(\mathbf{p})\mathbf{w} = 0$,

$\mathbf{G}(\tilde{\mathbf{p}}) = \frac{\partial \mathbf{g}(\tilde{\mathbf{p}})}{\partial \tilde{\mathbf{p}}} \mathbf{T}_p(\tilde{\mathbf{p}})$ is the product of the constraint Jacobian matrix $\frac{\partial \mathbf{g}}{\partial \mathbf{p}}$ and the generalized velocity transformation matrix $\mathbf{T}_p$.

Here $\mathbf{T}_p(\tilde{\mathbf{p}})$ is a block-diagonal matrix:

$$\mathbf{T}_p = diag(\mathbf{T}_{p,k}) \qquad \mathbf{T}_{p,k} = \frac{\partial \dot{\mathbf{p}}_k}{\partial \mathbf{w}_k}$$

Cline and Pai [CLI 03] showed that the pseudoinverse of $\mathbf{G}$ is definded, even when $\mathbf{G}\mathbf{G}^T$ is singular. They used a pseudoinverse formula based on a singular value decomposition (SVD) of $\mathbf{G}$. Thus, $\mathbf{G}^+$ is obtained by truncating the small (nearly zero) singular values. In [PRE 02] the code is published solving a pseudoinverse problem using SVD.

The calculation of $\mathbf{T}_p$ is trivial because each $\mathbf{T}_{p,k}$ can be calculated as the output parameter of the k-th constraint. But the computation of $\frac{\partial \mathbf{g}}{\partial \mathbf{p}}$ of the complete system is a challenge, because it may happen that the equation of a constraint does not only depend on generalized coordinates associated with this constraint, but also on other generalized coordinates.

Example 2.4. *Consider a 3-bodies closed loop system with revolute joints shown in Fig. 2.6. Here generalized coordinates are the relative angles* p_1, p_2, p_3 *associated with the first three constraints. Obviously, the derivative* $\frac{\partial \mathbf{g}_4}{\partial p_1}$ *depends on all* p_k (k=1...3)

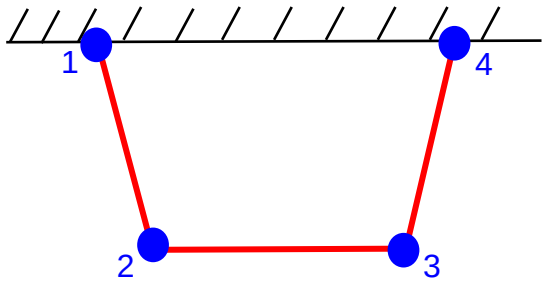

Figure 2.6: A 3-bodies closed loop system with revolute joints

That is why in the general case the derivative $\left(\frac{\partial \mathbf{g}_i}{\partial \mathbf{p}_1} \quad \cdots \quad \frac{\partial \mathbf{g}_i}{\partial \mathbf{p}_m} \right)$ can not be the output parameter of the i-th constraint object.

We can solve this problem if we use the equation:

$$\frac{\partial \mathbf{g}_i}{\partial \mathbf{p}_j} = \frac{\partial \mathbf{g}_i}{\partial \mathbf{q}_{\mathbf{B}_i}} \frac{\partial \mathbf{q}_{\mathbf{B}_i}}{\partial \mathbf{p}_j}$$

where $\mathbf{q}_{\mathbf{B}i}$ is the vector of coordinates of constraint's basic bodies. The matrix $\frac{\partial \mathbf{g}_i}{\partial \mathbf{q}_{\mathbf{B}_i}}$ can be set as the output parameters of the i-th constraint object.

Now we need to calculate the matrix $\frac{\partial \mathbf{q}_{\mathbf{B}_i}}{\partial \mathbf{p}_j}$. Consider an arbitrary body from $\boldsymbol{B}_i$. Let t denote the number of the body. There are two variants of the calculations of the deviation $\frac{\partial \mathbf{q}_t}{\partial \mathbf{p}_j}$. If the t-th body is dependent on the j-th constraint, then the deviation can be calculated from the output parameter $\frac{\partial \underline{q}_j(\mathbf{p}_j, \mathbf{q}_{\mathbf{B}_j})}{\partial \mathbf{p}_j}$ of the j-th constraint object (in Example 2.4 we calculated the partial derivative $\partial \mathbf{q}_3 / \partial p_3$ in this way).

The more sophisticated problem is to calculate the partial derivative $\frac{\partial \mathbf{q}_t}{\partial \mathbf{p}_j}$ when the t-th body depends on another constraint (e.g. $\partial \mathbf{q}_3 / \partial p_2$ in Example 2.4). Let the t-th body depends on the m-th constraint. Then $\frac{\partial \mathbf{q}_t}{\partial \mathbf{p}_j}$ is the part of the derivative $\frac{\partial \underline{q}_m(\mathbf{q}_{\mathbf{B}_\mathbf{m}}, \mathbf{p}_m)}{\partial \mathbf{p}_j}$, that can be calculated from the equation:

$$\frac{\partial \underline{q}_m(\mathbf{q}_{\mathbf{B}_\mathbf{m}}, \mathbf{p}_m)}{\partial \mathbf{p}_j} = \frac{\partial \underline{q}_m}{\partial \mathbf{q}_{\mathbf{B}_m}} \frac{\partial \mathbf{q}_{\mathbf{B}_m}}{\partial \mathbf{p}_j}$$

where $\frac{\partial \underline{q}_m}{\partial \mathbf{q}_{\mathbf{B}_m}}$ is the output parameter of the m-th constraint object. We should recursively repeat this routine for all bodies with coordinates included in $\mathbf{q}_{\mathbf{B}_m}$ and dependent on $\mathbf{p}_j$. In the recursion's end we obtain the case that was discussed before: we need to calculate the partial derivative $\frac{\partial \mathbf{q}_t}{\partial \mathbf{p}_j}$, where the t-th body is dependent on the j-th constraint.

Remark 2.6 *Implementing this routine in Example 2.4, we obtain:*

$$\frac{\partial \mathbf{q}_3}{\partial p_1} = \frac{\partial \underline{q}_3(\mathbf{q}_2, p_3)}{\partial p_1} = \frac{\partial \underline{q}_3}{\partial \mathbf{q}_2}\frac{\partial \mathbf{q}_2}{\partial p_1} = \frac{\partial \underline{q}_3}{\partial \mathbf{q}_2}\frac{\partial \underline{q}_2(\mathbf{q}_1, p_2)}{\partial p_1} = \frac{\partial \underline{q}_3}{\partial \mathbf{q}_2}\frac{\partial \underline{q}_2}{\partial \mathbf{q}_1}\frac{\partial \mathbf{q}_1}{\partial p_1} = \frac{\partial \underline{q}_3}{\partial \mathbf{q}_2}\frac{\partial \underline{q}_2}{\partial \mathbf{q}_1}\frac{\partial \underline{q}_1}{\partial p_1}$$

Finally, we obtain that the calculation of the global Jacobian matrix $\frac{\partial \mathbf{q}}{\partial \mathbf{p}}$ can be performed only using the partial derivatives $\frac{\partial \underline{q}_i}{\partial \mathbf{q}_{B_i}}$ and $\frac{\partial \underline{q}_i}{\partial \mathbf{p}_i}$ generated inside of constraint objects.

3 Computation Complexity

We estimate the complexity of the method using four basic estimations [GOL 93, PRE 02]:

1. The multiplication of two matrices **CB** where **C** is a [n,m] matrix and **B** is a [m,l] matrix **B** involves $O(n \cdot m \cdot l)$ floating point operations (multiplications and additions).

2. The complexity of the inversion of a [n,n] matrix is $O(n^3)$.

3. The pseudoinverse of a [m,n] matrix is $O(m \cdot n^2 + n^3)$ procedure.

4. Elimination of dependent rows from a [k,n] matrix has complexity $O(l \cdot n \cdot k)$, where $l = \min(k,n)$.

5. The complexity of eigendecomposition of a [n,n] matrix is $O(n^3)$.

Obviously, the time complexity of the simulation depends on many factors: the system's structure, the types of constraints, the number of joints and bodies, the number of processors, the structure of the hierarchy. In this chapter we calculate the complexities of basic subroutines that we execute during the simulation. Then we summarize them and obtain the method's complexity.

3.1 Stabilization complexity

Let us compare the complexity of post-stabilizations of absolute and generalized coordinates.

Consider a multibody system S. Let c denote the total number of constraints and n denote the total number of bodies in S. From (2.39) follows, that during the post-stabilization of absolute coordinates we need the pseudoinverse of two [$O(c)$, $O(n)$] matrices: $\frac{\partial \mathbf{g}}{\partial \mathbf{q}}$ and $\mathbf{G}$. Therefore, the post-stabilization of absolute coordinates has complexity $O(c \cdot n^2 + n^3)$.

Let us calculate the complexity of stabilization of generalized coordinates. The transformation from generalized to absolute coordinates and the backward transformation from generalized to absolute has complexity $O(c)$.

The more sophisticated problem is the calculation of the Jacobian matrix $\frac{\partial \mathbf{g}}{\partial \mathbf{p}}$ of the complete system. After the transformation to generalized coordinates the number of nonconfluent equations $\mathbf{g}$ reduces. Now $\mathbf{g}$ consists only of the equations of loop-closing constraints that are not included in the sequence of dependencies $\boldsymbol{C}$. For a system with t closed loops the number of equations in $\mathbf{g}$ is $O(t)$.

Let s_j denote the number of bodies in the j-th loop. Then the equation $\mathbf{g}_j=0$ of the loop-closing constraint depends on $O(s_j)$ generalized coordinates. The algorithm of the calculation $\frac{\partial \mathbf{g}_j}{\partial \mathbf{p}}$ has complexity $O(s_j)$. Obviously, the calculation of the global Jacobian matrix $\frac{\partial \mathbf{g}}{\partial \mathbf{p}}$ is $O(s)$ procedure, where $s=s_1+\ldots+s_t$ is the total number of bodies in loops.

Since the size of the Jacobian matrix $\frac{\partial \mathbf{g}}{\partial \mathbf{p}}$ is $[O(t),O(s)]$, it follows that the pseudoinverse of $\frac{\partial \mathbf{g}}{\partial \mathbf{p}}$ has complexity $O(s\cdot t^2+t^3)$.

Summing up all complexities, we obtain that the stabilization of generalized coordinates has complexity $O(s\cdot t^2+t^3)$, which is much less than the complexity of the stabilization of absolute coordinates.

3.2 Computation complexity of a basic subsystem

Consider a basic subsystem S consisting of n connected bodies. Let m denote the number of bodies that are connected by external joints with the complete system. Let c denote the number of internal constraints in $\mathbf{g}$. The Jacobian matrix $\mathbf{G}$ has size $[O(c), O(n)]$, the mass matrix $\mathbf{M}$ has size $[O(n),O(n)]$. Elimination of dependent rows from $\mathbf{G}$ has complexity $O(l\cdot n\cdot c)$, where $l=\min(c,n)$.

The matrix $\mathbf{G_e}$ has size [$O(c),O(m)$], $\mathbf{M_e}$ is a [$O(m), O(m)$] square matrix. Using equation (2.14), we obtain a complexity of $O(c\cdot n^2+c^3)$ for the calculation of $O(c)$-length vector $\mathbf{b}$ and [$O(c),O(m)$] matrix $\mathbf{S}$.

From (2.16) we obtain a complexity of $O(c\cdot m^2+ m^3)$ for the calculation of matrix $\mathbf{D}$ and a complexity of $O(c\cdot m+ m^3)$ for the calculation of vector $\mathbf{r}$.

From equation (2.17) we get a complexity of $O((n\text{-}m)^3+n\cdot c)$ for the calculation of accelerations of internal bodies $\dot{v}_i$.

Summing up all complexities, we obtain that on each time step the basic subsystem performs $O(n^3+c^3)$ calculations.

3.3 Computation complexity of a derived subsystem

Consider a derived subsystem S consisting of parents $S_1, S_2,\ldots,S_N$ connected by c constraints. Let n_{Ein} denote the number of internal bodies in S, and n_{Ext} denote the number of bordered bodies in S. The matrix $\hat{\mathbf{D}}$ from (2.22) is a square [$O(n), O(n)$] matrix, where $n=n_{Ext}+n_{Ein}$. The complexity of eigendecomposition of $\hat{\mathbf{D}}$ in (2.23) is $O(n^3)$.

The matrix $\mathbf{G}$ has size [$O(c), O(n)$]. From (2.24) follows, that the calculation $\tilde{\mathbf{G}}$ has complexity $O(c^3+c\cdot n^2)$.

The matrix $\hat{\mathbf{D}}'$ has size [$O(n), O(n_{Ext})$], $\tilde{\mathbf{G}}$ has size [$O(c), O(n)$], $\tilde{\mathbf{D}}$ is a [$O(n), O(n)$] square matrix, $\hat{\mathbf{r}}$ is an $O(n)$ vector. Using equation (2.29), we obtain a complexity $O(c\cdot n^2+c^3)$ for the calculation of the [$O(c),O(n_{Ext})$] matrix $\mathbf{S}$ and the $O(c)$-length vector $\mathbf{b}$.

From equation (2.31) we obtain a complexity of $O(c\cdot n^2)$ for the calculation of matrices $\mathbf{D}$ and $\mathbf{r}$.

If we know the vector $\boldsymbol{\tau}'$, then we can calculate the accelerations of internal bodies $\dot{\mathbf{v}}_{\mathbf{Ein}}$. The complexity of this calculation is $O(n\cdot c +n^2)$.

Summing up all complexities, we obtain that on each time step the basic subsystem performs $O(n^3+c^3)$ calculations.

3.4 Computation complexity of the method

Let us create the hierarchy of subsystems for a mechanical system S. Let all subsystems on all levels of the hierarchy have internal bodies and the number of bodies and internal constraints in each subsystem be limited by the global constant D. Therefore, the computation complexity of each subsystem is limited by $O(D^3)$.

Let n denote the total number of bodies in S. Since all subsystems have internal bodies, it follows that the total number of subsystems is limited by n.

Thus, we obtain that the global complexity of the computation of accelerations is $O(n \cdot D^3)$. Adding the complexity of the stabilization, we obtain that on each time step we perform $O(n \cdot D^3 + t^2 \cdot s + t^3)$ operations, where t is the number of closed loops in the system and s is the total number of bodies in loops.

4 Implementation Background

Trying to develop software for the simulation of dynamics of multibodies, we should remember that in a common way a multibody system is only a part of a sophisticated mechatronic system. Typically elements of mechatronic structures are electronic units, electromechanical transformers such as sensors, actors, pure data processing units as controllers and mechanical structures. These components are assembled according to their physical interfaces like mechanical connections, data transmissions, electric connections etc.

Our goal is to develop a tool that could be used for the simulation of mechanical parts of mechatronic systems. That is why we need that our software could be easily combined with electronic and control tools.

Our software is based on a strictly capsulated block-module concept [KAS 97]. In this context it means that the mechanical structure will be represented by separate objects which interact via predefined interfaces with each other. Using such interface, the objects could interact also with external software.

This approach has some significant advantages:

1. **Top-Down Design.** The design of the model structure can be done in a very physical-related manner. Models are partitioned in its physical units as they are constituted like a real system. The mechanical structure is kept as a particular component as it is connected in their real counterpart. The physical system borders will be kept in the virtual system as well. The model development can be performed in steps from a high grade of abstraction into more precise functionality. Changes in the topological structure or in the schematic (addition of sensors, etc.) will not affect the modelling procedure.

2. **Distributed Development.** The development of subsystems can be done at different places by various specialists. The global functionality can be assured by keeping the defined interfaces. There is no restriction which mathematical technique is used to describe the capsulated system behaviour as far as the interfaces will be maintained.

3. **Flexibility.** The model will get a high grade of exchangeability. Later developed, more complex and time consuming subsystems do not affect the development of other blocks and can be changed without degrading the performance of the whole model. The individual blocks can be tested separately. Once developed and tested modules can be reused in other applications. This will lead to an accumulation of block-models as a basis for block-oriented libraries to speed up development time for further modelling tasks.

4. **Quick Development.** An important precondition to retain good performance for this concept is a proper definition of the interfaces of the distinct blocks. Typically, the engineer is facing some particular problems. In order to get a clear arrangement of the distinct physical elements the separation of the real structure into the block elements has to be done in a physical- and design-related manner. Thus one obtains several model-blocks, each of them representing the corresponding mechanical subsystem.

Unlike of a huge number of other methods, we keep the block-module concept during the simulation. From a practical point of view, there are big advantages of a simulation on the basis of subsystems:

1. **Separate Testing.** Subsystems can be modelled, tested and compiled. Then they can be used in a way similar to software components that encapsulate their internal structure and can be connected via interfaces.

2. **Encapsulation of Critical Effects**. Critical effects like coulomb friction, non-permanent contacts etc. can be encapsulated inside a subsystem. While the changing of the structure of a subsystem we do not need to change the complete model structure.

3. **Distributed Simulation.** Subsystems are ideal candidates for the partitioning of large systems on multiple processors. During the simulation the main number of calculations proceeds inside of a subsystem. Therefore, we could easily distribute the simulation on several processors; each of them will work with its own subsystem.

Using this approach, we implemented our software in Visual Basic 6.0, but it can be easily partitioned in other existing object-oriented programming languages like Visual C, Delphi etc.

5 Basic Objects

In our method we split a simulating mechanical system into functional parts representing real components. Let us describe eight basic objects that are used in our algorithm. For the sake of simplicity we do not show the Visual Basic code, but only review the parameters and properties of objects.

Child objects describing the different types of constraints and forces are considered in the next chapter.

5.1 Timer

A timer object is used for the identification of the current time inside a simulating system. It has only two properties:

1. **Set the new current time value**.
2. **Show the current time value**.

During the calculations we use only one global time object. When we start a new integration step we set the new value of current time. All objects that need the current time value for their calculations (i.e. ground, force) get it from this timer.

5.2 Ground

We treat a ground object as a body whose motion is predefined. There is no restriction on the number of ground objects inside a simulating system. For example, inside a car model, described in Chapter 7, we use five ground objects.

While the simulation the object generates the following functions:

1. $\mathbf{q}[7,1]$ - Absolute coordinates.

$$\mathbf{q} = \left(\mathbf{x}^T \quad \boldsymbol{\theta}^T\right)^T$$

where

$\mathbf{x} = \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix}^T$ are Cartesian coordinates indicating the position of centre of mass of the body with respect to the global frame

$\boldsymbol{\theta} = \begin{pmatrix} e_0 & e_1 & e_2 & e_3 \end{pmatrix}^T$ is the vector of Euler parameters

2. $\mathbf{v}$[6,1] - Absolute velocity

$$\mathbf{v} = \begin{pmatrix} \dot{\mathbf{x}}^T & \boldsymbol{\Omega}^T \end{pmatrix}^T$$

where $\boldsymbol{\Omega} = \begin{pmatrix} \Omega_1 & \Omega_2 & \Omega_3 \end{pmatrix}^T$ is the global angular velocity vector.

3. $\dot{\mathbf{v}}$ [6,1] - Absolute acceleration.

4. $\mathbf{A}$[3,3] - Rotation matrix:

$$\mathbf{A} = \begin{pmatrix} e_0^2 + e_1^2 - e_2^2 - e_3^2 & 2(e_1 e_2 - e_0 e_3) & 2(e_1 e_3 + e_0 e_2) \\ 2(e_2 e_1 + e_0 e_3) & e_0^2 - e_1^2 + e_2^2 - e_3^2 & 2(e_2 e_3 - e_0 e_1) \\ 2(e_3 e_1 - e_0 e_2) & 2(e_3 e_2 + e_0 e_1) & e_0^2 - e_1^2 - e_2^2 + e_3^2 \end{pmatrix}$$

5. $\mathbf{T}$[7,6] - Velocity transformation matrix:

$$\dot{\mathbf{q}} = \mathbf{T}\mathbf{v}$$

6. $\overline{\mathbf{T}}$ [6,7] - Backward velocity transformation matrix:

$$\mathbf{v} = \overline{\mathbf{T}} \cdot \dot{\mathbf{q}}$$

The structure of matrices $\mathbf{T}$, $\overline{\mathbf{T}}$ was precisely described in Chapter 2, while the discussion of types of absolute coordinates.

5.3 Body

The difference between body objects and ground objects is that the motion of body objects is not predefined**.**

A body object has the following static parameters that should be set while the translation:

1. m - Mass.

2. $\bar{\mathbf{J}}$[3,3] - Moment of inertia expressed in the body frame connected to the centre of mass.

While the simulation we set the dynamical parameters of the object:

1. **q**[7,1] - Absolute coordinates.
2. **v**[6,1] - Absolute velocity.
3. $\dot{\mathbf{v}}$[6,1] - Absolute acceleration.
4. **f**[6,1] - Vector of external generalized forces acting on the body.

While the simulation the object generates the following functions:

1. **A**[3,3] - Rotation matrix:

$$\mathbf{A} = \begin{pmatrix} e_0^2 + e_1^2 - e_2^2 - e_3^2 & 2(e_1 e_2 - e_0 e_3) & 2(e_1 e_3 + e_0 e_2) \\ 2(e_2 e_1 + e_0 e_3) & e_0^2 - e_1^2 + e_2^2 - e_3^2 & 2(e_2 e_3 - e_0 e_1) \\ 2(e_3 e_1 - e_0 e_2) & 2(e_3 e_2 + e_0 e_1) & e_0^2 - e_1^2 - e_2^2 + e_3^2 \end{pmatrix}$$

2. **M**[6,6] - Mass matrix:

$$\mathbf{M} = \begin{pmatrix} m\mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{J} \end{pmatrix}$$

where

$\mathbf{I}_3$ is the [3,3] identity matrix,

$\mathbf{J} = \mathbf{A}\bar{\mathbf{J}}\mathbf{A}^T$ is the [3,3] moment of inertia matrix of the body expressed in the inertial frame.

3. **T**[7,6] - Velocity transformation matrix:

$$\dot{\mathbf{q}} = \mathbf{T}\mathbf{v}$$

4. $\overline{\mathbf{T}}$ [6,7] - Backward velocity transformation matrix:

$$\mathbf{v} = \overline{\mathbf{T}} \cdot \dot{\mathbf{q}}$$

5. g - Drift function of the normalization condition for Euler parameters:

$$g = e_0^2 + e_1^2 + e_2^2 + e_3^2$$

6. $\frac{\partial g}{\partial \mathbf{q}}$ [1,7] - Partial derivative of the drift:

$$\frac{\partial g}{\partial \mathbf{q}} = \begin{pmatrix} 0 & 0 & 0 & 2e_0 & 2e_1 & 2e_2 & 2e_3 \end{pmatrix}$$

5.4 Body output

Body outputs are interfaces used for connection of bodies with subsystem's external constraints.

Example 5.1. *Consider a subsystem shown in Fig. 5.1. From the physical point of view this is a system of two connected bodies. But from the object-oriented point of view this is a system consisting of two parents* S_1 *and* S_2 *connected by* $Constraint\ 1$.

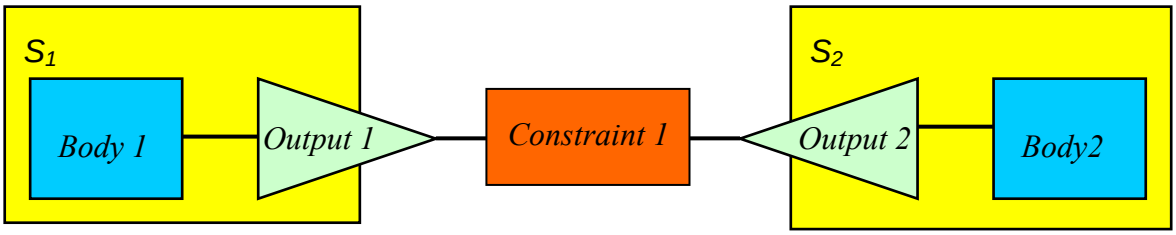

Figure 5.1: Two subsystems connected by the constraint

The typical way in object-oriented programming (e.g. Dymola software) is not to work with a body object outside of a body's subsystem but to create the special types of objects called *outputs* and to use them as bodies. This is possible because each output object (called *child*) inherits parameters (i.e. absolute coordinates, velocity) of its *parent* (a body or another output). If output's parameters change, then the output object automatically changes corresponding parameters of its parent.

This approach helps us to show explicitly on each level of hierarchy which bodies could be connected on the next level.

An output objects has the following static parameters that should be set while the translation:

1. *Parent* – Parent of the output (the body or the other output).

While the simulation the object inherits the same parameters of its parent:

1. $\mathbf{q}$[7,1] - Absolute coordinates,
2. $\mathbf{v}$[6,1] - Absolute velocity,
3. $\dot{\mathbf{v}}$[6,1] - Absolute acceleration,
4. $\mathbf{A}$[3,3] - Rotation matrix,
5. $\mathbf{T}$[7,6] - Velocity transformation matrix,
6. $\overline{\mathbf{T}}$ [6,7] - Backward velocity transformation matrix.

If the parent is not a ground object then the object inherits two more parameters from its parent:

1. $\mathbf{M}$[6,6] - Mass matrix,
2. $\mathbf{f}$[6,1] - Vector of external generalized forces acting on the parent.

The object has also its own parameter:

1. $\boldsymbol{\tau}$[6,1] - Generalized forces acting in links external to the output's subsystem.

Remark 5.1 *While the description of objects we do not make a difference between bodies and outputs* (*i.e. saying "The parameter of the constraint is an array of bodies" we mean that the parameters of the constraint is the array of body objects and output objects*)

5.5 Generalized force

A generalized force object describes an external force or external torque acting on bodies.

The object has the following static parameters that should be set while the translation:

1. $\boldsymbol{J}=\{Body\ j_1,\ Body\ j_2,\ldots,\ Body\ j_s\}$ - Array of bodies which are acted by the force, where s is the number of bodies in $\boldsymbol{J}$.

2. *Timer* - Timer object that provides the force object by the current time value.

While the simulation the object runs the subroutine:

1. **Applying the force**. Using the values of the coordinates $\mathbf{q}_\mathbf{J} = (\mathbf{q}_{j_1}^T \quad \ldots \quad \mathbf{q}_{j_s}^T)^T$ and velocities $\mathbf{v}_\mathbf{J} = \left(\mathbf{v}_{j_1}^T \quad \ldots \quad \mathbf{v}_{j_s}^T\right)^T$ of bodies in $\boldsymbol{J}$ and the current time t of the timer object, the force object calculates the vector $\mathbf{f}_\mathbf{J}$:

$$\mathbf{f}_\mathbf{J}(t,\ \ \mathbf{q}_\mathbf{J},\ \ \mathbf{v}_\mathbf{J}) = \left(\mathbf{f}_{j_1}^T \quad \ldots \quad \mathbf{f}_{j_s}^T\right)^T$$

   where t is the current time value, obtained from *Timer*.

   After this, the subroutine increases the parameter $\mathbf{f}$ of each body in $\boldsymbol{J}$ by the value of the correspondent element of $\mathbf{f}_\mathbf{J}$.

5.6 Constraint

A constraint object describes a holonomical constraint connecting several bodies.

The object has the following static parameters that should be set while the translation:

1. $\boldsymbol{J}$ - Array of bodies connected by the constraint,

2. $\boldsymbol{B}$ - Array of basic bodies, where $\boldsymbol{B} \subset \boldsymbol{J}$,

3. $\boldsymbol{K}$- Array of dependent bodies, where $\boldsymbol{K} \subset \boldsymbol{J}$.

If the constraint is included in the sequence of dependencies $\boldsymbol{C}$, then while the simulation we set the dynamical parameters of the object:

1. $\mathbf{p}$ - Generalized coordinates associated with the constraint,

2. $\mathbf{w}$ - Generalized velocities,

3. $\dot{\mathbf{w}}$ - Generalized acceleration,

4. $\mathbf{T}_p$ – Generalized velocity transformation matrix:

$$\dot{\mathbf{p}} = \mathbf{T}_p \mathbf{w}$$

While the simulation the object runs the subroutines:

1. **Set absolute coordinates of dependent bodies.** The constraint calculates the current value of the dependency function $\underline{q}(\mathbf{p}, \mathbf{q_B})$ and sets the current values of coordinates of dependent bodies:

$$\mathbf{q_K} := \underline{q}(\mathbf{p}, \mathbf{q_B})$$

Remark 5.2 *The operator ":=" indicates the changing of objects' parameters, e.g. expression "*$\mathbf{q_K} := \underline{q}$*" means that we set parameters* $\mathbf{q}$ *of all bodies in* $\boldsymbol{K}$ *equal the correspondent parts of vector function* $\underline{q}$*.*

2. **Set the absolute velocities of dependent bodies.** The constraint calculates the time derivative of the dependency function $\underline{v}(\mathbf{p}, \mathbf{q_B}, \mathbf{w}, \mathbf{v_B}) = \dot{\underline{q}}$ and sets the current value of the velocities of dependent bodies:

$$\mathbf{v_K} := \overline{\mathbf{T}}_\mathbf{K} \cdot \underline{v}(\mathbf{p}, \mathbf{q_B}, \mathbf{w}, \mathbf{v_B})$$

where $\overline{\mathbf{T}}_\mathbf{K}$ is the backward velocity transformation matrix:

$$\mathbf{v_K} = \overline{\mathbf{T}}_\mathbf{K} \cdot \dot{\mathbf{q}}_\mathbf{K} \qquad \overline{\mathbf{T}}_\mathbf{K} = \mathbf{diag}(\overline{\mathbf{T}}_i)$$

Here $\overline{\mathbf{T}}_i$ is the backward velocity transformation matrix of the i-th body in $\boldsymbol{K}$.

While the simulation the object generates the functions:

1. $\mathbf{g_1}(\mathbf{q_J})$ - Drift of the constraint for the absolute coordinates,

2. $\dfrac{\partial \mathbf{g_1}}{\partial \mathbf{q_J}}$ - Constraint Jacobian matrix,

3. $\mathbf{u}$ - Vector:

$$\mathbf{u} = \left[\frac{\mathbf{d}}{\mathbf{dt}} \left(\frac{\partial \mathbf{g}_1}{\partial \mathbf{q}_\mathbf{J}} \cdot \mathbf{T}_\mathbf{J} \right) \right] \cdot \mathbf{v}_\mathbf{J} \qquad \mathbf{T}_\mathbf{J} = \mathbf{diag}(\mathbf{T}_i)$$

Here $\mathbf{T}_i$ is the velocity transformation matrix of the *i*-th body in ***J***.

4. $\mathbf{g}_2(\mathbf{p})$ - Drift of the constraint for the generalized coordinates,

5. $\frac{\partial \mathbf{g}_2}{\partial \mathbf{p}}$ - Jacobian matrix,

6. $\frac{\partial \underline{q}(\mathbf{p}, \mathbf{q}_\mathbf{B})}{\partial \mathbf{p}}$ - Partial derivative of the dependency function $\underline{q}(\mathbf{p}, \mathbf{q}_\mathbf{B})$, a part of the Jacobian matrix,

7. $\frac{\partial \underline{q}(\mathbf{p}, \mathbf{q}_\mathbf{B})}{\partial \mathbf{q}_\mathbf{B}}$ - Partial derivative, a part of the Jacobian matrix,

8. $\underline{\varepsilon}$ - Dependency function, describing the relation between the generalized accelerations $\dot{\mathbf{w}}$ and the absolute coordinates $\mathbf{q}_\mathbf{J}$, velocities $\mathbf{v}_\mathbf{J}$, accelerations $\dot{\mathbf{v}}_\mathbf{J}$:

$$\dot{\mathbf{w}} = \underline{\varepsilon}(\mathbf{q}_\mathbf{J}, \mathbf{v}_\mathbf{J}, \dot{\mathbf{v}}_\mathbf{J}).$$

Remark 5.3 *For most types of joints (revolute, prismatic etc.) parameters* $\mathbf{g}_2(\mathbf{p})$ *and* $\frac{\partial \mathbf{g}_2}{\partial \mathbf{p}}$ *are equal to null since the equation of the constraint expressed in the generalized coordinates are singular. But for some types of constraints (e.g. ball joint) we do not have the singularity. We note that* $\mathbf{g}_1(\mathbf{q})$ *and* $\mathbf{g}_2(\mathbf{p})$ *can describe different drifts (e.g.* $\mathbf{g}_1(\mathbf{q})$ *of a ball joint object shows that the places of connection of both bodies coincide,* $\mathbf{g}_2(\mathbf{p})$ *of a ball joint object describes the normalization condition for Euler parameters of a dependent body). We precisely discuss this in Chapter 6 while the descriptions of the ball joint object.*

5.7 Basic subsystem

Basic subsystem object is a subsystem of the lowest level of hierarchy. It can include body objects, ground objects, force objects and output objects, but it can not include subsystems.

The object has the following static parameters that should be set while the translation:

1. $\boldsymbol{J}$ - Array of bodies in the subsystem,
2. $\boldsymbol{R}$ - Array of grounds in the subsystem,
3. $\boldsymbol{F}$ - Array of forces in the subsystem,
4. $\boldsymbol{C}$ - Array of constraints in the subsystem,
5. $\boldsymbol{O}$ - Array of outputs in the subsystem.

While the translation the object generates the static parameters:

1. $\boldsymbol{E}$ - Array of bordering bodies, where $\boldsymbol{E} \subset \boldsymbol{J}$ (obviously, $\boldsymbol{E}$ is equal to the array of bodies that have children in $\boldsymbol{O}$),
2. $\boldsymbol{I}$ - Array of internal bodies, where $\boldsymbol{I} \subset \boldsymbol{J}$ (obviously, $\boldsymbol{I}$ is equal to the array of bodies that do not have children in $\boldsymbol{O}$).

If the subsystem has external constraints, then while the simulation it generates the function:

1. $\mathbf{D}$, $\mathbf{r}$ - Dependency matrices that describe the relation between the accelerations $\dot{\mathbf{v}}_{\mathbf{E}}$ of bordering bodies and the forces $\boldsymbol{\tau}$ in external constraints:

$$\dot{\mathbf{v}}_{\mathbf{E}} = \mathbf{D}\boldsymbol{\tau} + \mathbf{r}$$

While the simulation the object runs the subroutine:

1. **Null forces.** The parameter $\mathbf{f}$ of all bodies from $\boldsymbol{J}$ is set equal to null,
2. **Apply forces.** All forces from $\boldsymbol{F}$ run the subroutine “Applying the force”,

3. **Set acceleration** (*when there are no external constraints*). The subsystem solves the equations of motion and calculates the current values of the accelerations $\dot{\mathbf{v}}_{\mathbf{J}}$. The parameter $\dot{\mathbf{v}}$ of each body in $\boldsymbol{J}$ is set equal to the value of the correspondent element of $\dot{\mathbf{v}}_{\mathbf{J}}$,

4. **Set internal acceleration** (*when there are external constraints*). The object obtains from the outputs in $\boldsymbol{O}$ the values of forces $\boldsymbol{\tau}$ in external constraints. Using $\boldsymbol{\tau}$ it calculates the new values of accelerations $\dot{\mathbf{v}}_{\mathbf{I}}$. The parameter $\dot{\mathbf{v}}$ of each body in $\boldsymbol{I}$ is set equal to the value of the correspondent element of $\dot{\mathbf{v}}_{\mathbf{I}}$.

5.8 Derived subsystem

Derived subsystem object is a subsystem of the high level of hierarchy. It can include other subsystems, ground objects, force objects and output objects. But it can not include body objects.

The object has the following static parameters that should be set while the translation:

1. $\boldsymbol{J} = \{S_1, \quad S_2, \quad \dots \quad S_N\}$ - Array of subsystem's parents,
2. $\boldsymbol{R}$ - Array of grounds in the subsystem,
3. $\boldsymbol{F}$ - Array of forces in the subsystem,
4. $\boldsymbol{C}$ - Array of constraints in the subsystem,
5. $\boldsymbol{O}$ - Array of outputs in the subsystem.

While the translation the object generates the static parameters:

1. $\boldsymbol{E}$ – Array of outputs of $S_1,\dots,S_N$: $\boldsymbol{E} = \bigcup_{i=1}^{N} \boldsymbol{O}(S_i)$,
2. $\boldsymbol{Ext}$ – Array of outputs bordering to the subsystem, where $\boldsymbol{Ext} \subset \boldsymbol{E}$ (obviously, $\boldsymbol{Ext}$ is equal to the array of outputs that have children in $\boldsymbol{O}$),

3. ***Ein*** – Array of outputs internal to the subsystem, where $\boldsymbol{Ein} \subset \boldsymbol{E}$ (obviously, ***Ein*** is equal to the array of outputs that do not have children in ***O***).

If the subsystem has external constraints, then while the simulation it generates the function:

1. $\mathbf{D}$, $\mathbf{r}$ - Dependency matrices which describe the relation between the accelerations $\dot{\mathbf{v}}_{\mathbf{Ext}}$ of bordering bodies and the forces $\boldsymbol{\tau}'$ in external constraints:

$$\dot{\mathbf{v}}_{\mathbf{Ext}} = \mathbf{D}\boldsymbol{\tau}' + \mathbf{r}$$

While the simulation the object runs the subroutines:

1. **Null forces.** All subsystems from ***J*** run the subroutine “Null forces”,

2. **Apply forces.** All forces from ***F*** run the subroutine “Applying the force”. All subsystems from ***J*** run the subroutine “Apply forces”,

3. **Set acceleration** (*when there are no external constraints*). Using the matrices $\mathbf{D}_{\boldsymbol{J}}$, $\mathbf{R}_{\boldsymbol{J}}$ of parents in ***J,*** the subsystem calculates the current values of accelerations $\dot{\mathbf{v}}_{\mathbf{Ein}}$ of objects in ***Ein*** and forces $\boldsymbol{\tau}_{\boldsymbol{Ein}}$, where $\boldsymbol{\tau}_{\boldsymbol{Ein}}$ are internal forces produced by constraints from ***C***. The parameters $\dot{\mathbf{v}}$ and $\boldsymbol{\tau}$ of each object in ***Ein*** are set equal to the correspondent elements of $\dot{\mathbf{v}}_{\mathbf{Ein}}$ and $\boldsymbol{\tau}_{\boldsymbol{Ein}}$ respectively,

4. **Set internal acceleration** (*when there are external constraints*). The object obtains from the outputs in ***O*** the values of forces $\boldsymbol{\tau}'$ in external constraints. Using $\boldsymbol{\tau}'$ it calculates the current values of accelerations $\dot{\mathbf{v}}_{\mathbf{Ein}}$ and forces $\boldsymbol{\tau}_{\boldsymbol{Ein}}$, where $\boldsymbol{\tau}_{\boldsymbol{Ein}}$ are internal forces produced by constraints from ***C***. The parameters $\dot{\mathbf{v}}$ and $\boldsymbol{\tau}$ of each object in ***Ein*** are set equal to the correspondent elements of $\dot{\mathbf{v}}_{\mathbf{Ein}}$ and $\boldsymbol{\tau}_{\boldsymbol{Ein}}$ respectively.

6 Components

In this chapter we show derived objects simulating the different types of constraints and forces. The description of other objects can be performed in a similar way.

6.1 Joints

We describe four most frequent types of joints: Revolute joint, Prismatic joint, Ball joint and Stiff connection. All of them are based on the constraint class (described in Chapter 5) and have all its parameters.

6.1.1 Revolute joint

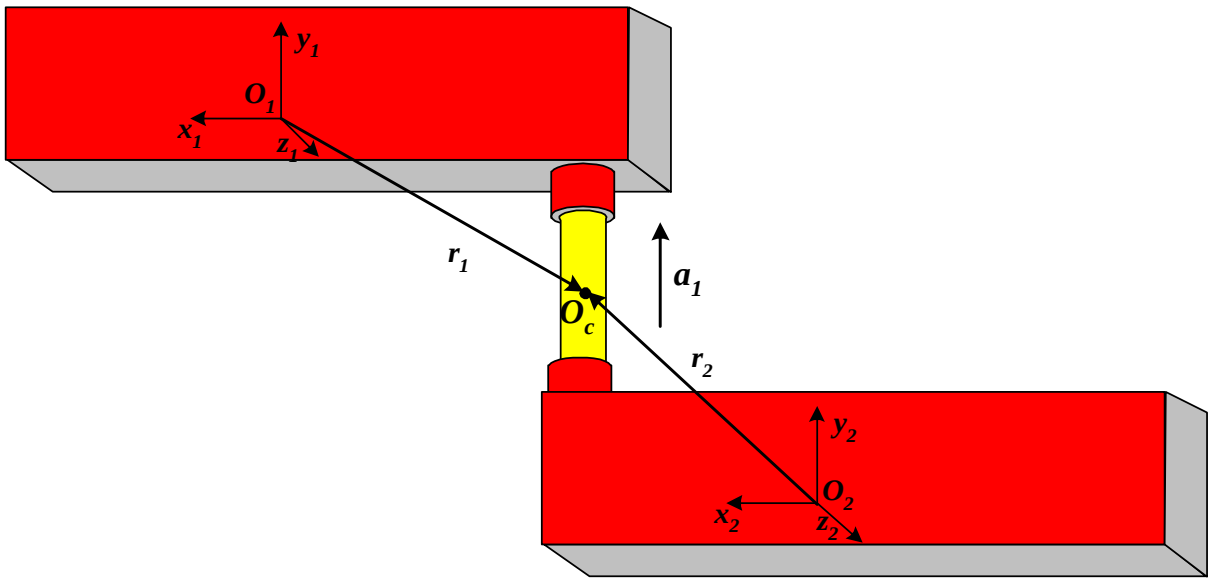

Figure 6.1: Revolute joint

Revolute joint object describes a revolute joint's connection of two bodies shown in Fig. 6.1. Let O_i be the centre of mass of *Body i* (i=1,2). Let O_c be the centre of the joint. By $O_i x_i y_i z_i$ denote the frame associated with the body. Let *Body 1* be basic and *Body 2* be dependent.

By $\mathbf{q_i} = \begin{pmatrix} \mathbf{x}_i^T & \boldsymbol{\theta}_i^T \end{pmatrix}^T$ denote the vector of position coordinates, where $\boldsymbol{x}_i = (x_{i,1}\ x_{i,2}\ x_{i,3})^T$ is the Cartesian coordinates of centre of mass of the body with respect to the global frame and $\boldsymbol{\theta}_i = \begin{pmatrix} e_{i,0} & e_{i,1} & e_{i,2} & e_{i,3} \end{pmatrix}^T$ are four Euler parameters indicating the orientation of the body. Let $\mathbf{A}_{0,i}$ be the rotation matrix of *Body i.*

The object has the following static parameters that should be set while the translation:

1. ***J***={*Body 1, Body 2*} - Array of bodies connected by the constraint,
2. ***B***={*Body 1*} - Array of basic bodies,
3. ***K***={*Body 2*} - Array of dependent bodies,
4. $\mathbf{r}_1[3,1]$ - *R*elative vector from O_1 to O_c, expressed in $O_1x_1y_1z_1$,
5. $\mathbf{r}_2[3,1]$ - Relative vector from O_2 to O_c, expressed in $O_2x_2y_2z_2$,
6. $\mathbf{a}_1[3,1]$ - Normalized axis of rotation expressed in $O_1x_1y_1z_1$.

While the translation the object generates the static parameter:

1. $\mathbf{a}_2[3,1]$ - Normalized axis of rotation expressed in $O_2x_2y_2z_2$

   Since the properties of rotation matrices, it follows that the absolute coordinates of $\mathbf{a}$ equal to $\mathbf{A}_{0,1}\mathbf{a}_1$. Thus, we can calculate $\mathbf{a}_2$ using initial conditions:

$$\mathbf{a}_2 = \mathbf{A}_{2,0}\big|_{t=0}\mathbf{A}_{0,1}\big|_{t=0}\mathbf{a}_1$$

   where $\mathbf{A}_{2,0} = \mathbf{A}_{0,2}^T$ is the backward rotation matrix of *Body 2*

If the joint is included in the sequence of dependencies ***C,*** then while the simulation we set the dynamical parameters of the object:

1. $p[1,1]$ - Generalized coordinate equal to the angle between the projections of $\mathbf{r}_1$ and $\mathbf{r}_2$ on the plane perpendicular to $\mathbf{a}$:

$$p = \arcsin\left(\frac{\left\|(\mathbf{A}_{0,1}\mathbf{a}_1 \times \mathbf{A}_{0,1}\mathbf{r}_1) \times (\mathbf{A}_{0,1}\mathbf{a}_1 \times \mathbf{A}_{0,2}\mathbf{r}_2)\right\|}{\left\|\mathbf{A}_{0,1}\mathbf{a}_1 \times \mathbf{A}_{0,1}\mathbf{r}_1\right\| \cdot \left\|\mathbf{A}_{0,1}\mathbf{a}_1 \times \mathbf{A}_{0,2}\mathbf{r}_2\right\|}\right)$$

2. w[1,1] - Generalized velocity equal to the time derivative of p:

$$w = \dot{p}$$

3. T_p=(1) - Generalized velocity transformation matrix*:*

$$\dot{p} = T_p w = w$$

4. $\dot{w}$ [1,1] - Generalized acceleration:

$$\dot{w} = \left(\dot{\mathbf{v}}_2 - \dot{\mathbf{v}}_1 - (\Omega_1 \times \mathbf{w})\right)^T \cdot \left(\mathbf{A}_{0,1}\mathbf{a}_1\right)$$

where

$\mathbf{w} = w \cdot \mathbf{A}_{0,1}\mathbf{a}_1$ is the vector of relative angular velocity.

$\mathbf{\Omega}_1$ is the global angular velocity of *Body 1*

5. $\mathbf{s}$[4,1] - Euler parameters describing the relative rotation around the axis $\mathbf{a}_1$:

$$s = (s_0 \quad \mathbf{s}^T)^T = (\cos(p/2) \quad \sin(p/2)\mathbf{a}_1^T)^T$$

While the simulation the object runs the subroutines:

1. **Set absolute coordinates of the dependent body.** The constraint calculates the current value of the dependency function $\underline{q}$:

$$\underline{q}(p, \mathbf{q}_1) = \begin{pmatrix} \mathbf{x_1} + \mathbf{A}_{0,1}\mathbf{r}_1 - \mathbf{A}_{0,1}\mathbf{A}_{1,2}(\mathbf{s})\mathbf{r}_2 \\ \boldsymbol{\theta} \end{pmatrix}$$

where

$\mathbf{A}_{1,2}(\mathbf{s})$ is the matrix of relative rotation,

$\boldsymbol{\theta}=\boldsymbol{\theta}_1 \circ \mathbf{s}$ is the vector of Euler parameters describing the rotation of *Body 2*.

After calculation of $\underline{q}$ the object sets the current values of the coordinates of the dependent body:

$$Body\ 2.\mathbf{q} := \underline{q}$$

2. **Set the absolute velocity of the dependent body.** The object calculates the time derivative $\underline{v}(p, \mathbf{q}_1, w, \mathbf{v}) = \dot{\underline{q}}$ and sets the current values of the absolute velocity of the dependent body:

$$Body\ 2.\mathbf{v} := \overline{\mathbf{T}}_2 \cdot \underline{v}(p, \mathbf{q_1}, w, \mathbf{v_1})$$

where $\overline{\mathbf{T}}_2$ is the backward velocity transformation matrix of *Body 2***.**

Now consider the functions generated by the object on each time step:

1. $\mathbf{g}_1$[6,1] - Drift of the constraint for the absolute coordinates:

$$\mathbf{g}_1 = \begin{pmatrix} (\mathbf{x}_1 + \mathbf{A}_{0,1}\mathbf{r}_1) - (\mathbf{x}_2 + \mathbf{A}_{,,2}\mathbf{r}_2) \\ \mathbf{A}_{0,1}\mathbf{a}_1 - \mathbf{A}_{0,2}\mathbf{a}_2 \end{pmatrix}$$

2. $\frac{\partial \mathbf{g}_1}{\partial \mathbf{q}}$ [6,14] - Constraint Jacobian matrix,

3. $\mathbf{u}$[6,1] - Vector:

$$\mathbf{u} = \left[\frac{\mathbf{d}}{\mathbf{dt}} \left(\frac{\partial \mathbf{g}_1}{\partial \mathbf{q}_\mathbf{J}} \cdot \mathbf{T}_\mathbf{J} \right) \right] \cdot \mathbf{v}_\mathbf{J} \qquad \mathbf{T}_\mathbf{J} = \mathbf{diag}(\mathbf{T}_1, \mathbf{T}_2)$$

where $\mathbf{T}_i$ is the velocity transformation matrix of the i-th body,

4. $g_2(p) \equiv 0$ - Drift of the constraint for the generalized coordinate,

5. $\frac{\partial g_2}{\partial p} \equiv 0$ - Derivative of $g_2(p)$,

6. $\frac{\partial \underline{q}(p, \mathbf{q}_1)}{\partial p}$ [7,1] - Partial derivative,

7. $\frac{\partial \underline{q}(p, \mathbf{q}_1)}{\partial \mathbf{q}_1}$ [7,7] - Partial derivative.

6.1.2 Prismatic joint

Prismatic joint object describes a prismatic joint's connection of two bodies shown in Fig. 6.2. We implemented the object as the combination of a prismatic joint and a revolute joint because bodies can slide and rotate and along the axis $\mathbf{a}_1$. For the sake of simplicity we do not mention the joint's parameters associated with the rotation because they are absolutely the same as revolute joint's parameters.

Let O_i be the centre of mass of *Body i* (i=1,2). Let M_i be the place of connection of the body. By $O_i x_i y_i z_i$ denote the frame associated with the body. Let *Body 1* be basic and *Body 2* be dependent.

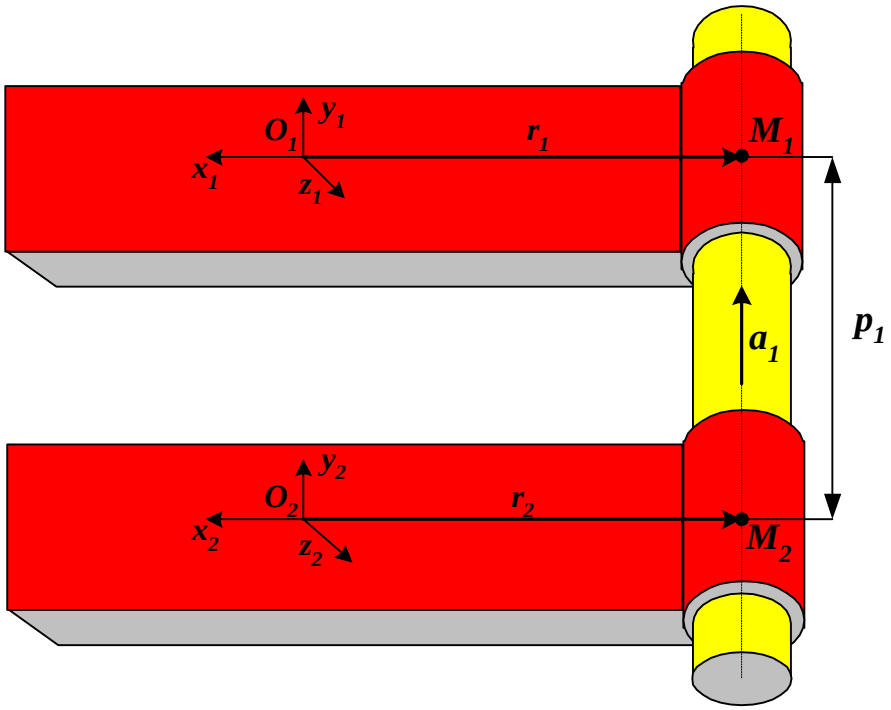

Figure 6.2: Prismatic joint

The object has the following static parameters that should be set while the translation:

1. $\boldsymbol{J} = \{Body\ 1, Body\ 2\}$ - Array of bodies connected by the constraint,
2. $\boldsymbol{B} = \{Body\ 1\}$ - Array of basic bodies,
3. $\boldsymbol{K} = \{Body\ 2\}$ - Array of dependent bodies,
4. $\mathbf{r}_1$[3,1] - Relative vector from O_1 to M_1, expressed in $O_1x_1y_1z_1$,
5. $\mathbf{r}_2$[3,1] - Relative vector from O_2 to M_2, expressed in $O_2x_2y_2z_2$,
6. $\mathbf{a}_1$[3,1] - Normalized axis of sliding expressed in $O_1x_1y_1z_1$.

While the translation the object generates the static parameter:

1. $\mathbf{a}_2$[3,1] - Normalized axis of rotation expressed in $O_2x_2y_2z_2$. It can be calculated using initial values:

$$\mathbf{a}_2 = \mathbf{A}_{2,0}\big|_{t=0}\,\mathbf{A}_{0,1}\big|_{t=0}\,\mathbf{a}_1$$

If the joint is included in the sequence of dependencies $\boldsymbol{C}$, then while the simulation we set the dynamical parameters of the object:

1. p[1,1] - Generalized coordinate equal to the projection of distance between M_1 and M_2 on the axis $\mathbf{a}$:

$$p = \left(\mathbf{A}_{0,1}\mathbf{a}_1\right)^T \cdot \left(\mathbf{x}_2 + \mathbf{A}_{0,2}\mathbf{r}_2 - \mathbf{x}_1 - \mathbf{A}_{0,1}\mathbf{r}_1\right)$$

2. w[1,1] - Generalized velocity equal to the time derivative of p:

$$w = \dot{p}$$

3. T_p=(1) - Generalized velocity transformation matrix:

$$\dot{p} = T_p w = w$$

4. $\dot{w}$ [3,1] - Generalized acceleration.

While the simulation the object runs the subroutines:

1. **Set the absolute coordinates of the dependent body.** The constraint calculates the current value of the dependency function $\underline{q}$:

$$\underline{q}(p,\mathbf{q}_1)=\begin{pmatrix}\mathbf{x}_1+\mathbf{A}_{0,1}\mathbf{r}_1+p\cdot\mathbf{A}_{0,1}\mathbf{a}_1-\mathbf{A}_{0,1}\mathbf{r}_2\\ \boldsymbol{\theta}_1\end{pmatrix}$$

After calculation of $\underline{q}$ the object sets the current values of coordinates of dependent body:

$$Body\ 2.\mathbf{q}:=\underline{q}$$

2. **Set the absolute velocity of the dependent body.** The object calculates the time derivative $\underline{v}(p,\mathbf{q}_1,w,\mathbf{v})=\dot{\underline{q}}$ and sets the current values of the absolute velocity of the dependent body:

$$Body\ 2.\mathbf{v}:=\overline{\mathbf{T}}_2\cdot\underline{v}(p,\mathbf{q_1},w,\mathbf{v_1})$$

where $\overline{\mathbf{T}}_2$ is the backward velocity transformation matrix of *Body 2***.**

Now consider the functions generated by the object on each time step:

1. $\mathbf{g}_1$[6,1] - Drift of the constraint for the absolute coordinates:

$$\mathbf{g}_1=\begin{pmatrix}\mathbf{A}_{0,1}\mathbf{a}_1\times(\mathbf{x}_2+\mathbf{A}_{0,2}\mathbf{r}_2-\mathbf{x}_1-\mathbf{A}_{0,1}\mathbf{r}_1)\\ \mathbf{A}_{0,1}\mathbf{a}_1-\mathbf{A}_{0,2}\mathbf{a}_2\end{pmatrix}$$

2. $\frac{\partial\mathbf{g_1}}{\partial\mathbf{q}_1}$ [6,14] - Constraint Jacobian matrix

3. $\mathbf{u}$[6,1] - Vector:

$$\mathbf{u}=\left[\frac{\mathbf{d}}{\mathbf{dt}}\left(\frac{\partial\mathbf{g}_1}{\partial\mathbf{q_J}}\cdot\mathbf{T_J}\right)\right]\cdot\mathbf{v_J}\qquad\mathbf{T_J}=\mathbf{diag}(\mathbf{T}_1,\mathbf{T}_2)$$

where $\mathbf{T}_i$ is the velocity transformation matrix of the i-th body,

4. $g_2(p)\equiv 0$ - Drift of the constraint for the generalized coordinate,

5. $\frac{\partial g_2}{\partial p} \equiv 0$ - Derivative of $g_2(p)$,

6. $\frac{\partial \underline{q}(p, \mathbf{q}_1)}{\partial p}$ [7,1] - Partial derivative,

7. $\frac{\partial \underline{q}(p, \mathbf{q}_1)}{\partial \mathbf{q}_1}$ [7,7] - Partial derivative.

6.1.3 Ball joint

Ball joint object describes the ball joint's connection of two bodies shown in Fig. 6.3.

Let O_c be the centre of the joint. Let O_i be the centre of mass of *Body i* (i=1,2). By $O_i x_i y_i z_i$ denote the frame associated with the body. Let *Body 1* be basic and *Body 2* be dependent.

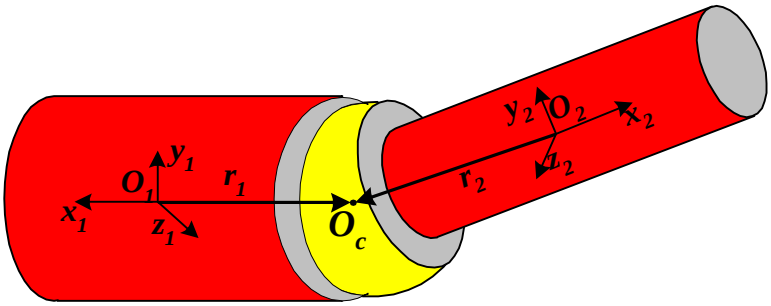

Figure 6.3: Ball joint

The object has the following static parameters that should be set while the translation:

1. ***J*** = {*Body 1, Body 2*} - Array of bodies connected by the constraint,

2. ***B*** = {*Body 1*} - Array of basic bodies,

3. ***K*** = {*Body 2*} - Array of dependent bodies,

4. $\mathbf{r}_1$[3,1] - Relative vector from O_1 to M_1, expressed in $O_1x_1y_1z_1$,

5. $\mathbf{r}_2$[3,1] - Relative vector from O_2 to M_2, expressed in $O_2x_2y_2z_2$.

If the joint is included in the sequence of dependencies ***C,*** then while the simulation we set the dynamical parameters of the object:

1. $\mathbf{p}=(p_1 \;\; p_2 \;\; p_3 \;\; p_4)^T$ - Vector of generalized coordinates equal to the vector of Euler parameters of the dependent body:

$$\mathbf{p} = \boldsymbol{\theta}_2$$

2. $\mathbf{w}=(w_1 \;\; w_2 \;\; w_3)^T$ - Vector of generalized velocity equal to the global angular velocity vector of the dependent body:

$$\mathbf{w} = \boldsymbol{\Omega}_2$$

3. $\mathbf{T}_p$ - Generalized velocity transformation matrix:

$$\dot{\mathbf{p}} = \mathbf{T}_p\mathbf{w} = \left[\frac{1}{2}\begin{pmatrix} -p_2 & p_1 & -p_4 & p_3 \\ -p_3 & p_4 & p_1 & -p_2 \\ -p_4 & -p_3 & p_2 & p_1 \end{pmatrix}^T\right]\mathbf{w}$$

4. $\dot{\mathbf{w}}$ - Generalized acceleration.

While the simulation the object runs the subroutines:

1. **Set absolute coordinates of the dependent body.** The constraint calculates the current value of the dependency function $\underline{q}$:

$$\underline{q}(\mathbf{p},\mathbf{q}_1) = \begin{pmatrix} \mathbf{x}_1 + \mathbf{A}_{0,1}\mathbf{r}_1 - \mathbf{A}_{0,1}\mathbf{A}_{1,2}(\mathbf{p})\mathbf{r}_2 \\ \mathbf{p} \end{pmatrix}$$

   where $\mathbf{A}_{1,2}(\mathbf{p})$ is the relative rotation matrix.

   After calculation of $\underline{q}$ the object sets the new values of the absolute coordinates of the dependent body:

$$Body\ 2.\mathbf{q} := \underline{q}$$

2. **Set the absolute velocity of the dependent body.** The object calculates the time derivative $\underline{v}(p, \mathbf{q}_1, w, \mathbf{v}) = \dot{\underline{q}}$ and sets the current values of the absolute velocity of the dependent body:

$$Body\ 2.\mathbf{v} := \overline{\mathbf{T}}_2 \cdot \underline{v}(\mathbf{p}, \mathbf{q}_1, \mathbf{w}, \mathbf{v}_1)$$

where $\overline{\mathbf{T}}_2$ is the backward velocity transformation matrix of *Body 2***.**

Now consider the functions generated by the object on each time step:

1. $\mathbf{g}_1$[3,1] - Drift of the constraint for the absolute coordinates:

$$\mathbf{g}_1 = (\mathbf{x}_2 + \mathbf{A}_{0,2}\mathbf{r}_2) - (\mathbf{x}_1 + \mathbf{A}_{0,1}\mathbf{r}_1)$$

2. $\frac{\partial \mathbf{g}_1}{\partial \mathbf{q}}$ [3,14] - Constraint Jacobian matrix,

3. $\mathbf{u}$[3,1] - Vector:

$$\mathbf{u} = \left[\frac{\mathbf{d}}{\mathbf{dt}}\left(\frac{\partial \mathbf{g}_1}{\partial \mathbf{q}_\mathbf{J}} \cdot \mathbf{T}_\mathbf{J}\right)\right] \cdot \mathbf{v}_\mathbf{J} \qquad \mathbf{T}_\mathbf{J} = \mathbf{diag}(\mathbf{T}_1, \mathbf{T}_2)$$

where $\mathbf{T}_i$ is the velocity transformation matrix of the *i*-th body,

4. $g_2(\mathbf{p})$[1,1] - Drift of the constraint for the generalized coordinates:

$$g_2(\mathbf{p}) = p_1^2 + p_2^2 + p_3^2 + p_4^2$$

5. $\frac{\partial g_2(\mathbf{p})}{\partial \mathbf{p}}$ [4,1] - Derivative of $g_2(\mathbf{p})$

$$\frac{\partial g_2}{\partial \mathbf{p}} = \begin{pmatrix} 2p_1 & 2p_2 & 2p_3 & 2p_4 \end{pmatrix}$$

6. $\frac{\partial \underline{q}(\mathbf{p}, \mathbf{q}_1)}{\partial p}$ [7,4] - Partial derivative,

7. $\frac{\partial \underline{q}(\mathbf{p},\mathbf{q}_1)}{\partial \mathbf{q}_1}$ [7,7] - Partial derivative.

6.1.4 Stiff connection

Stiff connection describes a rigid connection of two bodies shown in Fig. 6.4.

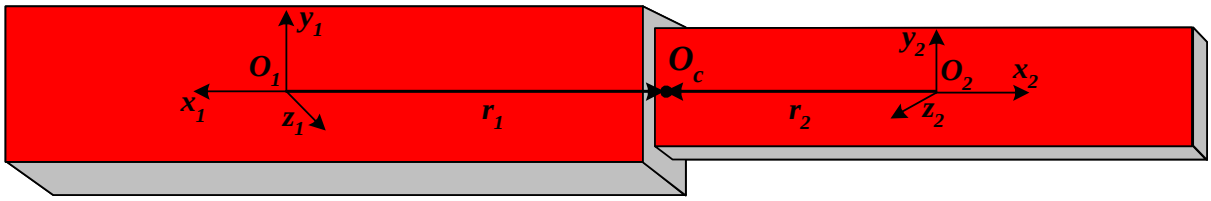

Figure 6.4: Stiff connection

Let O_c be the centre of the joint. Let O_i be the centre of mass of *Body i* (*i*=1,2). By $O_i x_i y_i z_i$ denote the frame associated with the body. Let *Body 1* be basic and *Body 2* be dependent.

The object has the following static parameters that should be set while the translation:

1. ***J*** = {*Body 1*, *Body 2*} - Array of bodies connected by the constraint,
2. ***B*** = {*Body 1*} - Array of basic bodies,
3. ***K***= {*Body 2*} - Array of dependent bodies,
4. $\mathbf{r}_1$[3,1] - Relative vector from O_1 to M_1, expressed in $O_1 x_1 y_1 z_1$,
5. $\mathbf{r}_2$[3,1] - Relative vector $\mathbf{r}_2$ from O_2 to M_2, expressed in $O_2 x_2 y_2 z_2$,
6. $\mathbf{s}=(s_0\ s_1\ s_2\ s_3)^T$ - Euler parameters describing the relative rotation from $O_1 x_1 y_1 z_1$ to $O_2 x_2 y_2 z_2$.

While the translation the object generates the static parameter:

1. $\mathbf{A}_{1,2}(\mathbf{s})$[3,3] - Matrix of relative rotation:

$$\mathbf{A}_{1,2}(\mathbf{s}) = \begin{pmatrix} s_0^2 + s_1^2 - s_2^2 - s_3^2 & 2(s_1 s_2 - s_0 s_3) & 2(s_1 s_3 + s_0 s_2) \\ 2(s_2 s_1 + s_0 s_3) & s_0^2 - s_1^2 + s_2^2 - s_3^2 & 2(s_2 s_3 - s_0 s_1) \\ 2(s_3 s_1 - s_0 s_2) & 2(s_3 s_2 + s_0 s_1) & s_0^2 - s_1^2 - s_2^2 + s_3^2 \end{pmatrix}$$

Since the properties of rotation matrices, it follows:

$$\mathbf{A_{0,2}} = \mathbf{A_{0,1}} \mathbf{A_{1,2}}(\mathbf{s})$$

Obviously, the joint does not have generalized coordinates. While the simulation the object runs the subroutines:

1. **Set absolute coordinates of the dependent body.** The constraint calculates the current value of the dependency function $\underline{q}$:

$$\underline{q}(\mathbf{q_1}) = \begin{pmatrix} \mathbf{x_1} + \mathbf{A}_{0,1}\mathbf{r_1} - \mathbf{A}_{0,1}\mathbf{A}_{1,2}(\mathbf{s})\mathbf{r_2} \\ \boldsymbol{\theta} \end{pmatrix}$$

   where $\boldsymbol{\theta}=\boldsymbol{\theta}_1 \circ \mathbf{s}$ is the vector of Euler parameters describing the rotation of Body_2.

   After calculation of $\underline{q}$ the object sets the current values of the coordinates of the dependent body:

$$Body\ 2.\mathbf{q} := \underline{q}$$

2. **Set the absolute velocity of the dependent body.** The object calculates the time derivative $\underline{v}(p, \mathbf{q}_1, w, \mathbf{v}) = \dot{\underline{q}}$ and sets the current values of the absolute velocity of the dependent body:

$$Body\ 2.\mathbf{v} := \overline{\mathbf{T}}_2 \cdot \underline{v}(\mathbf{q}_1, \mathbf{v}_1)$$

   where $\overline{\mathbf{T}}_2$ is the backward velocity transformation matrix of *Body 2***.**

Now consider the functions generated by the object on each time step:

1. $\mathbf{g_1}$[7,1] - Drift of the constraint for the absolute coordinates:

$$\mathbf{g_1} = \begin{pmatrix} (\mathbf{x_2} + \mathbf{A_{0,2}}\mathbf{r_2}) - (\mathbf{x_1} + \mathbf{A_{0,1}}\mathbf{r_1}) \\ \boldsymbol{\theta}_2 - \boldsymbol{\theta}_1 \circ \mathbf{s} \end{pmatrix}$$

2. $\frac{\partial \mathbf{g_1}}{\partial \mathbf{q}_1}$ [7,14] - Constraint Jacobian matrix,

3. **u**[7,1] - Vector:

$$\mathbf{u} = \left[\frac{\mathbf{d}}{\mathbf{dt}} \left(\frac{\partial \mathbf{g}_1}{\partial \mathbf{q_J}} \cdot \mathbf{T_J} \right) \right] \cdot \mathbf{v_J} \qquad \mathbf{T_J} = \mathbf{diag}(\mathbf{T}_1, \mathbf{T}_2)$$

where $\mathbf{T}_i$ is the velocity transformation matrix of the *i*-th body,

4. $\frac{\partial \underline{q}(\mathbf{q}_1)}{\partial \mathbf{q}_1}$ [7,7] - Partial derivative.

6.2 Forces

We describe three frequent types of generalised forces: Gravity force, Spring Damper, and Cosine torque. All of them are based on Generalized force class described in Chapter 5.

6.2.1 Gravity force

A gravity force object simulates the impact of the gravity force on bodies.

The object has the following static parameters that should be set while the translation:

1. ***J***={*Body 1, Body 2,..., Body s*} - Array of bodies that includes all bodies in a simulating system,

2. g=9.8 - Gravity constant,

3. $\mathbf{e}=(e_1\ e_2\ e_3)^T$ - Gravity direction.

While the simulation the object runs the subroutine:

1. **Applying the force.** For each *Body i* in ***J*** the object calculates the current value of the gravity force acting on the body:

$$\mathbf{f}_i = gm_i \cdot \begin{pmatrix} e_1 & e_2 & e_3 & 0 & 0 & 0 \end{pmatrix}^T \qquad i = 1,...,s$$

where m_i is the mass of $Body\ i$.

Then the object increases the parameter **f** of the body:

$$Body\ i.\mathbf{f} := Body\ i.\mathbf{f} + \mathbf{f}_i \qquad i = 1,...,s$$

6.2.2 Spring damper

Spring damper object simulates a spring with a damper between two bodies shown in Fig. 6.5. Let O_i be the centre of mass of $Body\ i$ (i=1,2). Let M_i be the place of connection of the body. By $O_i x_i y_i z_i$ denote the frame associated with the body.

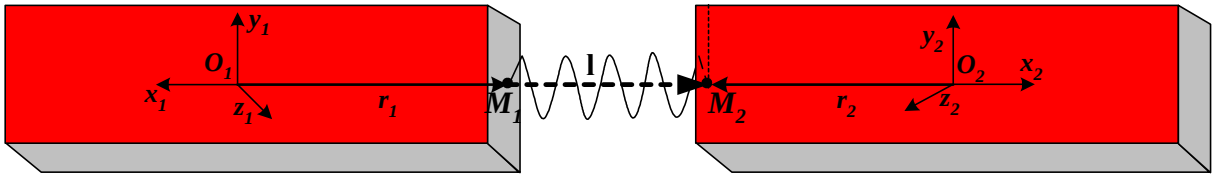

Figure 6.5: Spring damper

The object has the following static parameters that should be set while the translation:

1. ***J*** = {$Body\ 1$, $Body\ 2$} - Array of bodies connected by the constraint,
2. $\mathbf{r}_1$[3,1] - Relative vector from O_1 to M_1, expressed in $O_1 x_1 y_1 z_1$,
3. $\mathbf{r}_2$[3,1] - Relative vector from O_2 to M_2, expressed in $O_2 x_2 y_2 z_2$,
4. k - Spring constant,
5. l_0 - Unstretched length,
6. c - Damping constant,

7. δ - Accuracy.

While the simulation the object runs the subroutine:

1. **Applying the force.** The object calculates the current value of the spring force:

$$\mathbf{f} = \begin{cases} \left(k(|\mathbf{l}| - l_0) + c\left|\frac{d\mathbf{l}}{dt}\right| \right) \cdot \mathbf{e} & when \quad |\mathbf{l}| \geq \delta \\ (0 \quad 0 \quad 0)^T & when \quad |\mathbf{l}| < \delta \end{cases}$$

where

$\mathbf{l}$ is the length of the spring:

$$\mathbf{l} = (\mathbf{x}_2 + \mathbf{A}_{0,2}\mathbf{r}_2) - (\mathbf{x}_1 + \mathbf{A}_{0,1}\mathbf{r}_1)$$

$\mathbf{e}$ is the normalized direction of the force:

$$\mathbf{e} = \frac{\mathbf{l}}{|\mathbf{l}|}$$

and the spring torques:

$$\mathbf{t}_1 = (\mathbf{A}_{0,1}\mathbf{r}_1) \times \mathbf{f}$$
$$\mathbf{t}_2 = -(\mathbf{A}_{0,2}\mathbf{r}_2) \times \mathbf{f}$$

Then the object increases the parameter $\mathbf{f}$ of connected bodies:

$$Body\ 1.\mathbf{f} := Body\ 1.\mathbf{f} + \left(\mathbf{f}^T \quad \mathbf{t}_1{}^T\right)^T$$
$$Body\ 2.\mathbf{f} := Body\ 2.\mathbf{f} + \left(-\mathbf{f}^T \quad \mathbf{t}_2{}^T\right)^T$$

6.2.3 Cosine torque

A cosine torque object simulates the torque acting on a body, where the torque's value is the cosine waves. The direction of the torque is fixed in body’s frame.

Let O_1 be the centre of mass of the body. By $O_1x_1y_1z_1$ denote the frame associated with the body.

The object has the following static parameters that should be set while the translation:

1. $\boldsymbol{J} = \{Body\ 1\}$ - Array of bodies,
2. $\mathbf{a}_1[3,1]$ - Direction of the torque expressed in $O_1x_1y_1z_1$,
3. C - Amplitude,
4. k - Frequency,
5. $Timer$ - Timer object.

While the simulation the object runs the subroutine:

1. **Applying the force.** The object calculates the current value of the torque:

$$\mathbf{t} = C \cdot \cos(2\pi \cdot k \cdot Timer.t) \cdot \mathbf{A}_{\mathbf{0,1}}\mathbf{a}_{\mathbf{1}}$$

   Then the object increases the parameter $\mathbf{f}$ of the body:

$$Body\ 1.\mathbf{f} := Body\ 1.\mathbf{f} + \begin{pmatrix} 0 & 0 & 0 & \mathbf{t}^T \end{pmatrix}^T$$

7 Car Example

To validate the method presented in the preceding chapters, we have performed a number of calculations for the problem of a car system shown in Fig. 7.1.

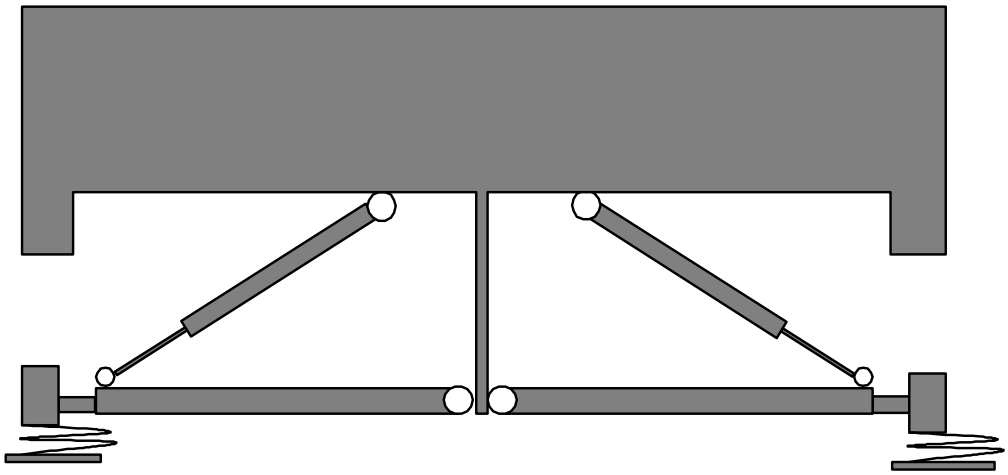

Figure 7.1: Car system

The complete system consists of several subsystems: Damper, Wheel, Suspension. This example perfectly illustrates all advantages of our method: the object-oriented simulation of multibodies, the stabilization of closed-loop system, the numerical efficiency of the combination of absolute and generalized coordinates.

All values of parameters are expressed in SI units: lengths – in meters, masses – in kilograms, etc. For notational simplicity we do not mention them while the model's description.

In future we will always say "The vector from body" meaning the vector from the body's centre of mass.

7.1 Wheel Subsystem

From the physical point of view Wheel Subsystem shown in Fig. 7.2 describes a wheel connected with a ground by a spring. It is a Basic Subsystem consisting of a few objects: *Ring*, *Ground*, *Spring* and *Wheel Output*, where *Wheel Output* is needful for the descriptions of constraints, that include the wheel on next steps of hierarchy.

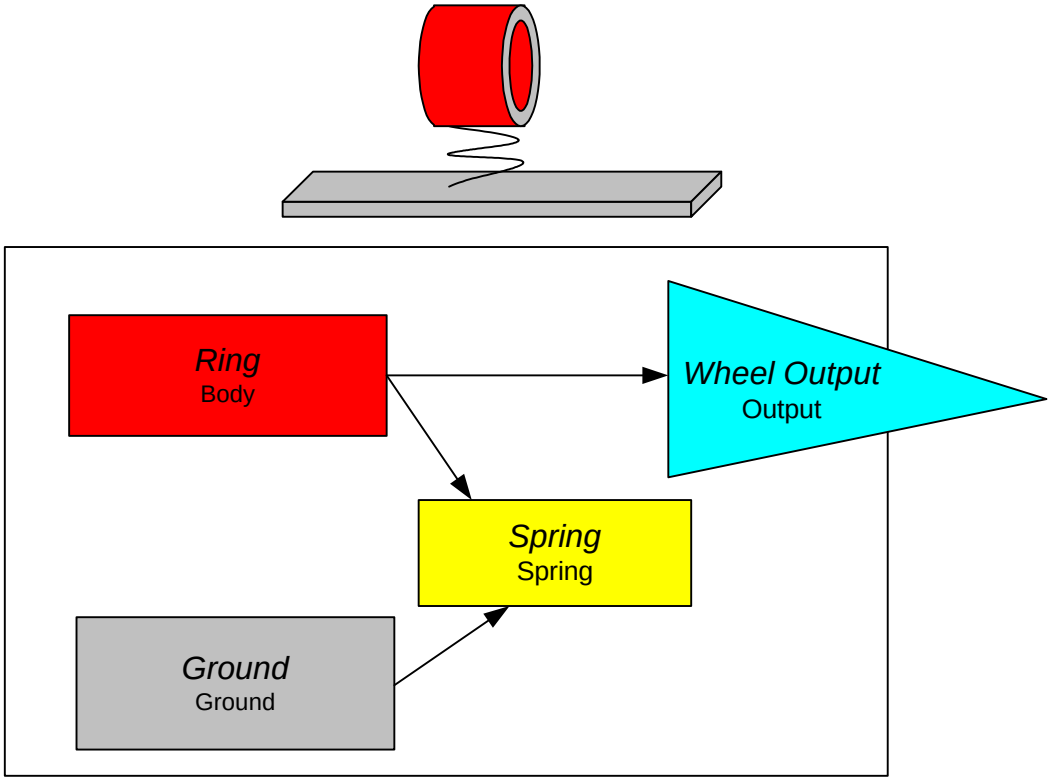

Figure 7.2: Wheel Subsystem

While the description of the subsystem we use the following parameters:

7.1.1 *Spring* parameters

1. $\boldsymbol{J}$={*Ground, Wheel*} - Array of connected bodies,
2. $\mathbf{r}_1=(0\ 0\ 0)^T$- Distance from the ground to the ground's place of connection,
3. $\mathbf{r}_2=(0\ 0\ 0)^T$- Distance from the wheel to the wheel's place of connection,
4. $k=4\cdot10^5$ - Spring constant,
5. $l_0=0.35$ - Outstretched length,
6. $c=100$ - Damping constant,
7. $\delta=10^{-13}$ - Accuracy.

7.1.2 *Ring* parameters

1. m=15 - Mass,
2. h=0.3 - Height,
3. r=0.3 - Radius,
4. Δh=0.01 - Width of the wheel.

7.2 Beam Subsystem

Since Derived Subsystem objects cannot include body objects we make the universal Basic Subsystem object. Beam Subsystem shown in Fig. 7.3 can be included in derived subsystems of next levels of hierarchy. The subsystem consists of *Beam* and the beam's child *Beam Output*.

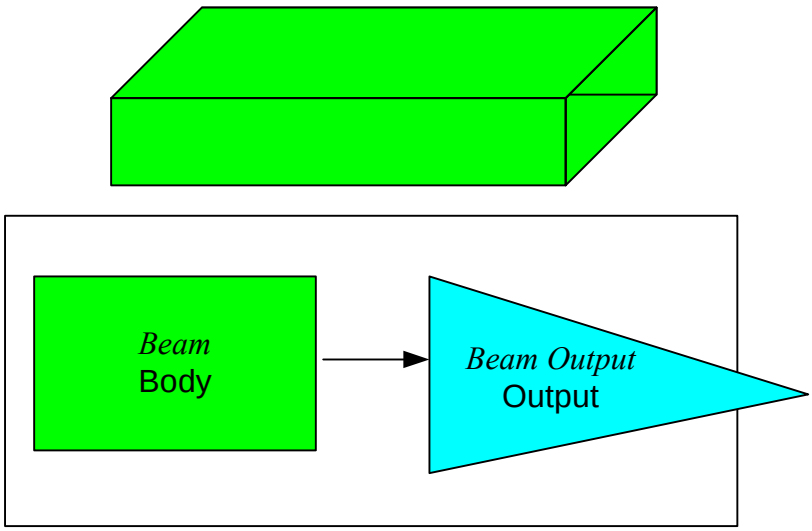

Figure 7.3: Beam Subsystem

While the description of the subsystem we use the following parameters:

1. m - Mass,
2. $\bar{\mathbf{J}}$[3,3] - Moment of inertia expressed in *Beam* frame connected with the centre of mass.

7.3 Damper Subsystem

From the physical point of view the damper shown in Fig. 7.4 is a mechanical subsystem consisting of a cylinder and a piston connected by a spring and by a prismatic joint.

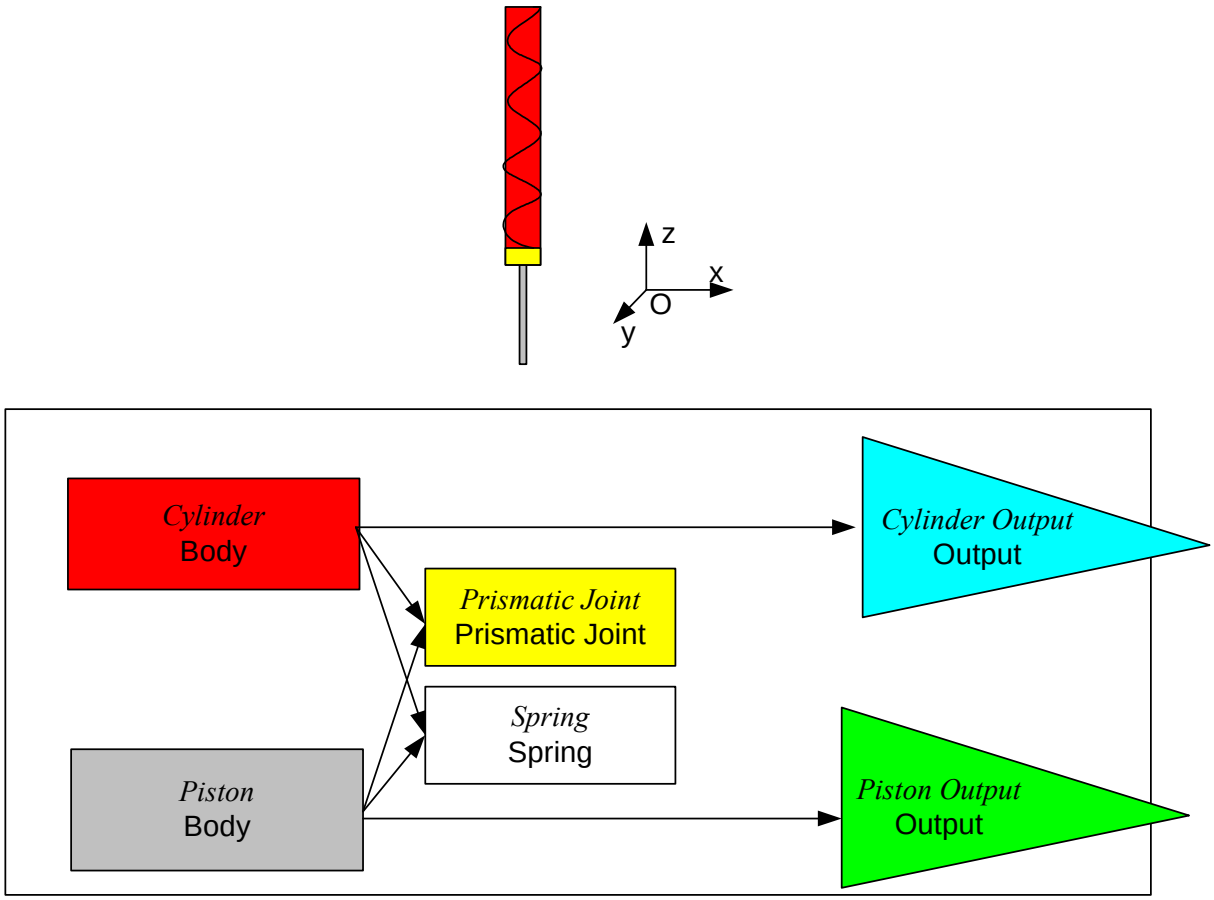

Figure 7.4: Damper Subsystem

From the object-oriented point of view Damper Subsystem is a Basic Subsystem consisting of two Body objects (*Cylinder* and *Piston*), *Prismatic Joint*, *Spring* and two Output objects (*Cylinder Output* and *Piston Output*).

While the description of the subsystem we use the following parameters:

7.3.1 *Spring* parameters

1. $\boldsymbol{J}$={*Piston, Cylinder*} - Array of connected bodies,
2. $\mathbf{r}_1$=(0 0 -0.3)^T- Distance from the piston to the piston's place of connection,
3. $\mathbf{r}_2$=(0 0 -0.3)^T- Distance from the cylinder to the cylinder's place of connection,
4. k=5·10^4 - Spring constant,
5. l_0=0.35 - Unstretched length,
6. c=10^4 - Damping constant.

7.3.2 *Cylinder* parameters

1. m=3.4 - Mass,
2. h=0.6 - Height,
3. r=0.03 - Radius,
4. Δh=0.004 - Width of the wall.

7.3.3 *Piston* parameters

1. m=13.23 - Mass,
2. h=0.6 - Height,
3. r=0.01 - Radius.

7.3.4 *Prismatic joint* parameters

1. $\boldsymbol{J}$={*Cylinder, Piston*} - Array of connected bodies,
2. $\boldsymbol{B}$={*Piston*} - Array of basic bodies,
3. $\boldsymbol{K}$={*Cylinder*} - Array of dependent bodies,
4. $\mathbf{r}_1$=(0 0 -0.3)^T - Distance from the cylinder to the joint,

5. $\mathbf{r}_2=(0\ 0\ -0.3)^T$ - Distance from the piston to the joint,

6. $\mathbf{a}_1=(0\ 0\ 1)^T$ - Normalized axis of sliding.

7.4 Suspensions Subsystem

From the physical point of view Suspension Subsystem shown in Fig. 7.5 is a subsystem consisting of a damper and a beam, where damper's piston is connected with the beam by a revolute joint (the axis of the joint is perpendicular to the frontal plane).

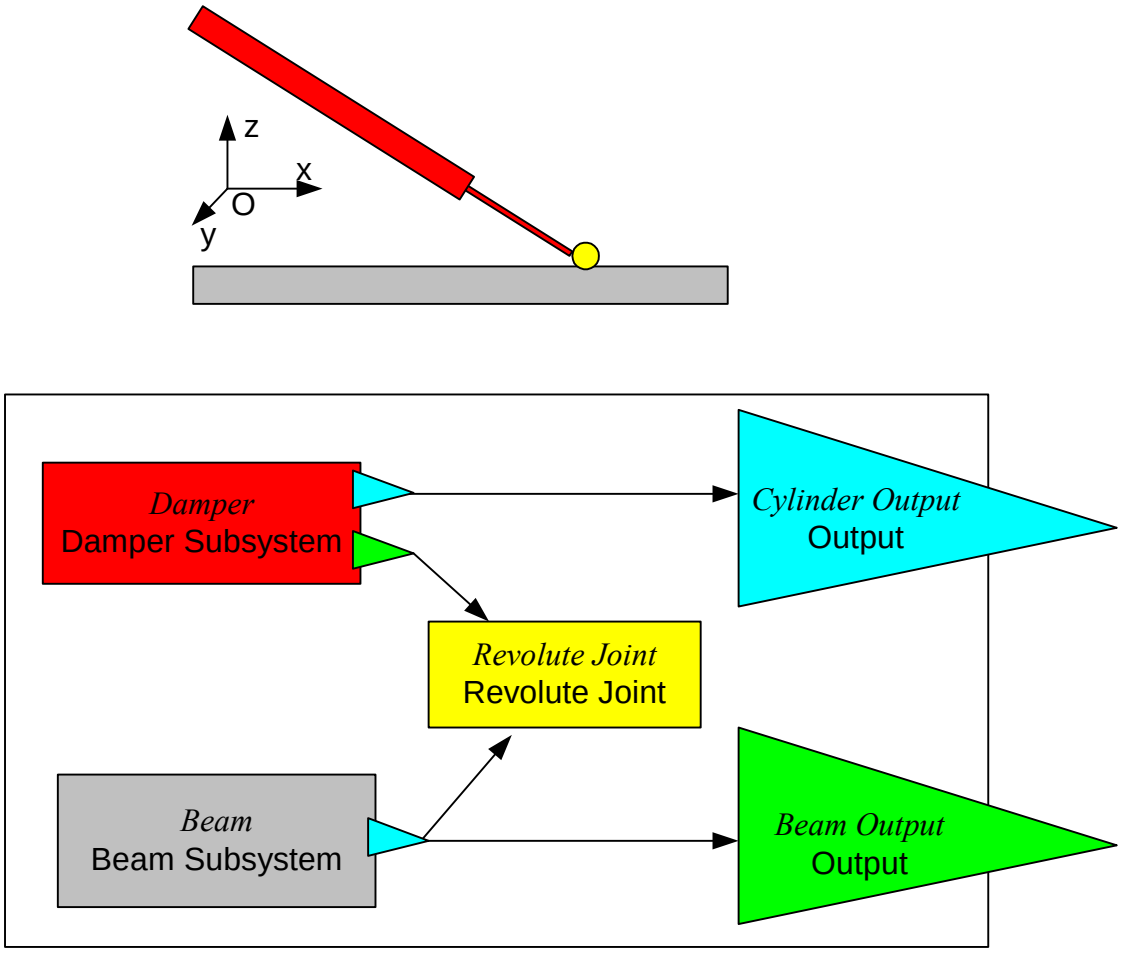

Figure 7.5: Suspension subsystem

From the object-oriented point of view Suspension Subsystem is a Derived Subsystem consisting of *Damper*, *Beam*, *Revolute Joint*, and two Outputs (*Cylinder Output* and *Beam Output*).

In our model we use left and right types of suspensions. The left suspension is obtained from the right by the horizontal rotation. While the description of the left suspension we use the following parameters:

7.4.1 *Beam* parameters

1. m=7.02 - Mass,
2. $\bar{\mathbf{J}}$=*diag*(0.06, 0.5855265, 0.5855265) - Moment of inertia.

7.4.2 *Revolute joint* parameters

1. ***J***={*Beam.Beam Output, Damper.Piston Output*} - Array of connected bodies,
2. ***B***={*Beam.Beam Output*} - Array of basic bodies,
3. ***K***={*Damper.Piston Output*} - Array of dependent bodies,
4. $\mathbf{r}_1=(0\ 0\ 0.2)^T$ - Distance from the beam to the joint,
5. $\mathbf{r}_2=(-0.3\ 0\ 0)^T$ - Distance from the piston to the joint,
6. $\mathbf{a}_1=(0\ 1\ 0)^T$ - Axis of rotation.

7.5 Car with suspension

From the physical point of view Car System shown in Fig. 7.6 consists of a car body connected with two suspensions by revolute joints with y-axis of rotation and two wheels connected with suspensions by revolute joints with x-axis of rotation. Trying to prevent the model from moving away, we connected the car body with a ground by the prismatic joint with z-axis of sliding.

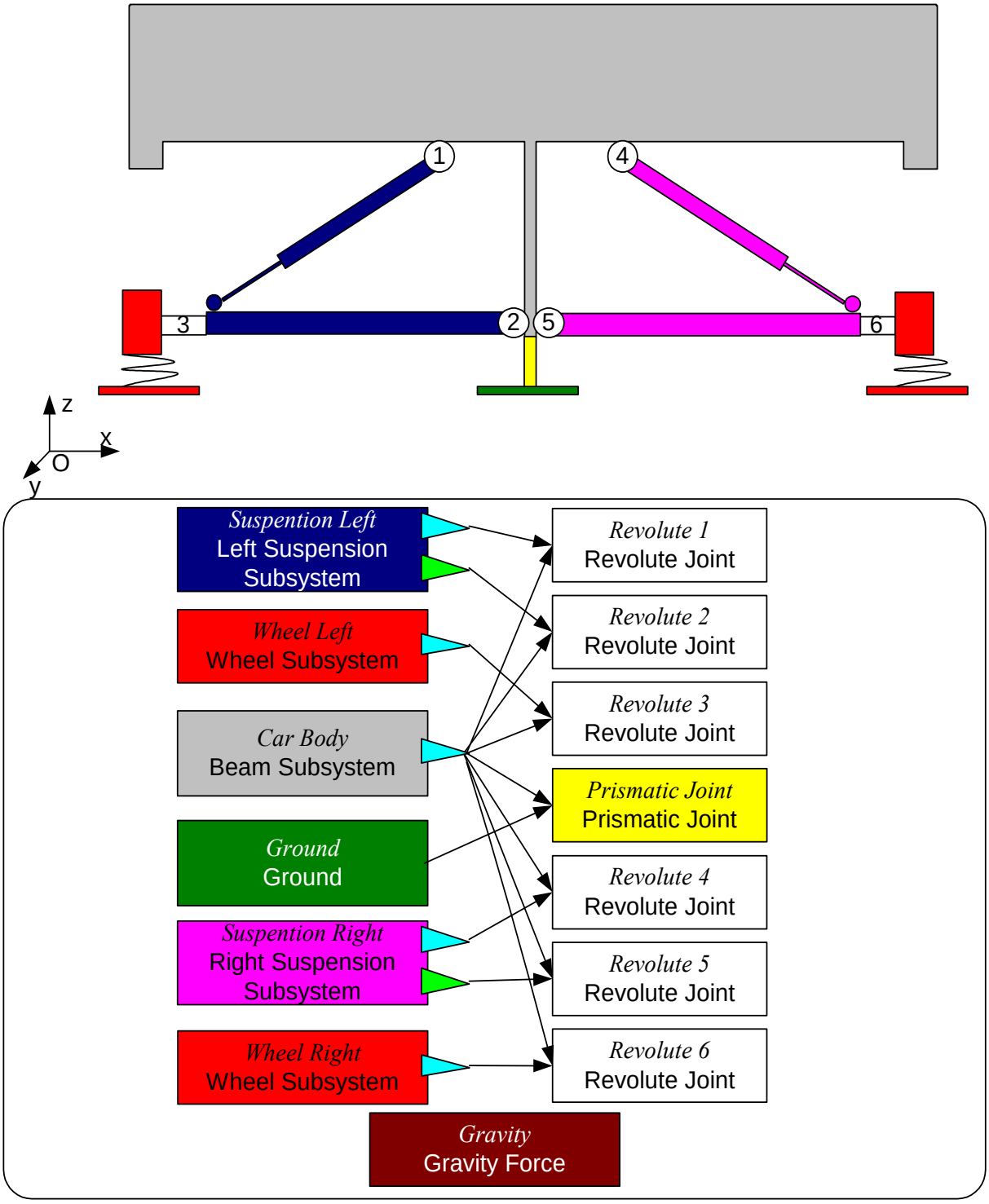

Figure 7.6: Car System

From the object-oriented point of view Car System is a Derived Subsystem consisting of Beam Subsystem (*Car Body*), two Wheel Subsystems (*Wheel Left* and *Wheel Right*), *Right Suspension*, *Left Suspension*, six Revolute Joint objects, *Ground* and *Gravity*.

While the description of Car System we use the following parameters:

7.5.1 *Car Body* parameters

1. m=585 - Mass,
2. $\bar{\mathbf{J}}$ = *diag*(12.675, 450.9375, 439.2375) - Moment of inertia.

7.5.2 *Revolute 1* parameters

1. ***J***={*Car Body.Beam Output, Suspension Left.Cylinder Output*} - Array of connected bodies,
2. $\mathbf{r}_1$ =(-0.1 0 0)^T - Distance from the car body to the joint,
3. $\mathbf{r}_2$ =(0 0 0.3)^T - Distance from the cylinder to the joint,
4. $\mathbf{a}_1$=(0 1 0)^T - Axis of rotation.

7.5.3 *Revolute 2* parameters

1. ***J***={*Car Body.Beam Output, Suspension Left.Beam Output*} - Array of connected bodies,
2. ***B***={*Car Body.Beam Output*} - Array of basic bodies,
3. ***K***={*Suspension Left.Beam Output*} - Array of dependent bodies,
4. $\mathbf{r}_1$=(0 0 -0.6)^T - Distance from the car body to the joint,
5. $\mathbf{r}_2$=(0.5 0 0)^T - Distance from the beam to the joint,
6. $\mathbf{a}_1$=(0 1 0)^T - Axis of rotation.

7.5.4 *Revolute 3* parameters

1. ***J***={*Suspension Left.Beam Output, Wheel Left.Wheel Output*} - Array of connected bodies,

2. ***B***={*Suspension Left.Beam Output* } - Array of basic bodies,
3. ***K***={*Wheel Left.Wheel Output* } - Array of dependent bodies,
4. $\mathbf{r}_1$ =(-0.5 0 0)^T - Distance from the beam to the joint,
5. $\mathbf{r}_2$=(0 0 0)^T - Distance from the wheel to the joint,
6. $\mathbf{a}_1$=(1 0 0)^T - Axis of rotation.

7.5.5 *Revolute 4* parameters

1. ***J***={*Car Body.Beam Output, Suspension Right.Cylinder Output*} - Array of connected bodies,
2. $\mathbf{r}_1$=(0.1, 0, 0)^T - Distance from the car body to the joint,
3. $\mathbf{r}_2$=(0, 0, 0.3)^T - Distance from the cylinder to the joint,
4. $\mathbf{a}_1$=(0, 1, 0)^T - Axis of rotation.

7.5.6 *Revolute 5* parameters

1. ***J***={*Car Body.Beam Output, Suspension Right.Beam Output*} - Array of connected bodies,
2. ***B***={*Car Body.Beam Output*} - Array of basic bodies,
3. ***K***={*Suspension Right.Beam Output*} - Array of dependent bodies,
4. $\mathbf{r}_1$=(0 0 -0.6)^T - Distance from the car body to the joint,
5. $\mathbf{r}_2$=(-0.5 0 0)^T - Distance from the beam to the joint,
6. $\mathbf{a}_1$=(0 1 0)^T - Axis of rotation.

7.5.7 *Revolute 6* parameters

1. ***J***={*Suspension Right.Beam Output, Wheel Right.Wheel Output*} - Array of connected bodies,

2. $\boldsymbol{B}$={*Suspension Right.Beam Output*} - Array of basic bodies,
3. $\boldsymbol{K}$={*Wheel Right.Wheel Output*} - Array of dependent bodies,
4. $\mathbf{r}_1$ =(-0.5 0 0)^T - Distance from the beam to the joint,
5. $\mathbf{r}_2$=(0 0 0)^T - Distance from the wheel to the joint,
6. $\mathbf{a}_1$=(1 0 0)^T - Axis of rotation.

7.5.8 *Prismatic joint* parameters

7. $\boldsymbol{J}$={*Ground, Car Body.Beam Output*} - Array of connected bodies,
8. $\boldsymbol{B}$={*Ground*} - Array of basic bodies,
9. $\boldsymbol{K}$={*Car Body.Beam Output*} - Array of dependent bodies,
10. $\mathbf{r}_1$ =(0 0 0)^T - Distance from the ground to the joint,
11. $\mathbf{r}_2$=(0 0 -0.95)^T - Distance from the car body to the joint,
12. $\mathbf{a}_1$=(0 0 1)^T - Axis of sliding.

7.5.9 *Gravity* parameters

1. g=9.8 - Free fall acceleration,
2. $\mathbf{e}$=(0 0 -1)^T - Gravity direction.

7.6 Array of independent bodies and sequence of dependencies

The array of independent bodies $\boldsymbol{I}$ is null. Three ground objects: *Car System.Ground*, *Wheel Left.Ground, Wheel Right.Ground* are marked in Fig. 7.7 by red.

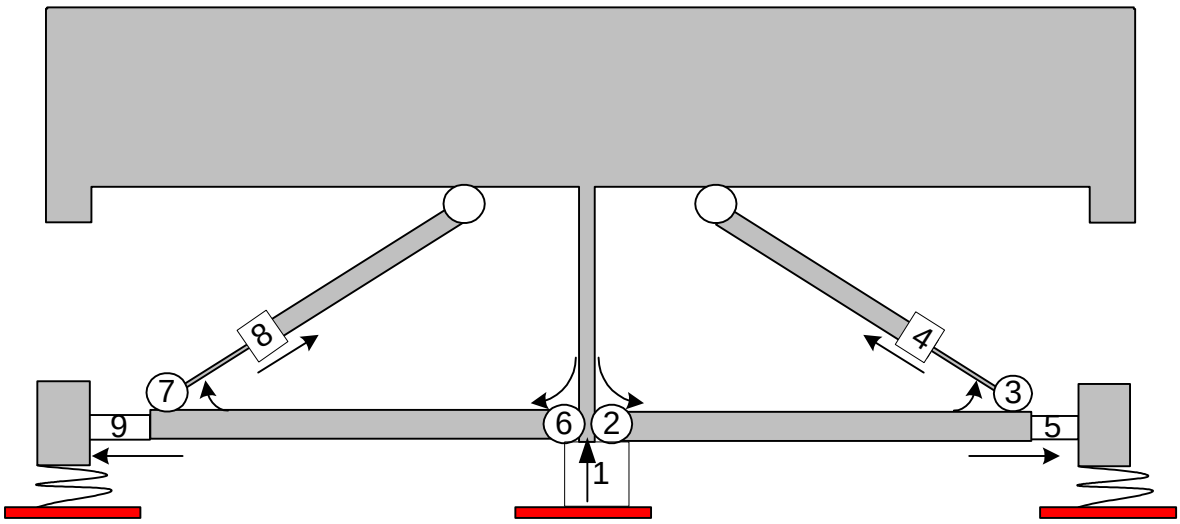

Figure 7.7: Ground objects and Sequence of dependencies

The constraints numbers in Fig. 7.7 are the order numbers of constraints in the sequence of dependencies ***C***={*Car System. Prismatic Joint, Car System.Revolute 5, Suspension Right.Revoulte Joint, Suspension Right.Damper.Prismatic Joint, Car System.Revolute 6, Car System.Revolute 2, Suspension Left.Revolute Joint, Suspension Left.Damper.Prismatic Joint, Car System.Revolute 3*}. The loop-closing constraints *Car System.Revolute 1* and *Car System.Revolute 4* are not included in ***C***.

Arrows in the figure show the way of calculation of absolute coordinates of dependent bodies.

Because of the closed structure of the model, we have the drift problem in the constraints *Car System.Revolute 1* and *Car System.Revolute 4*.

7.7 Start values

The start value of the 12-th length vector of generalized coordinates is:

1-2. Distance parameter p_1=0 and the angle of rotation p_2=0 of *Prismatic Joint*,

3. Angle of rotation p_1=0 of *Car System.Revolute 5*,

4. Angle of rotation p_1=-0.7854 of *Suspension Right.Revoulte Joint*,

5-6. Distance parameter p_1=0.2485 and the angle of rotation p_2=0 of *Suspension Right.Damper.Prismatic Joint*,

7. Angle of rotation p_1=0 of *Car System.Revolute 6*,

8. Angle of rotation p_1=0 of *Car System.Revolute 2*,

9. Angle of rotation p_1=0.7854 of *Suspension Left.Revoulte Joint*,

10-11. Distance parameter p_1=0.2485 and the angle of rotation p_2=0 of *Suspension Left.Damper.Prismatic Joint*,

12. The angle of rotation p_2=0 of *Car System.Revolute 3*.

The start value of 12-th length vector of generalized velocities is equal to null.

7.8 Simulation data

We perform the simulation of the model using the method described in Chapter 2. The simulation data shows the numerical efficiency and stability of our solution.

We choose the time interval to be [0,1.5]. Simulation was performed with Runge-Kutta method of the fourth order with the fixed time step equal to 0.001 *s*.

In Fig. 7.8-7.12 is shown the dynamics of *Car Body* and *Wheel Left* together with the drift of the system. The simulation data shows that the algorithm is stable and the model's drift is constant and has the order of the computation accuracy

.

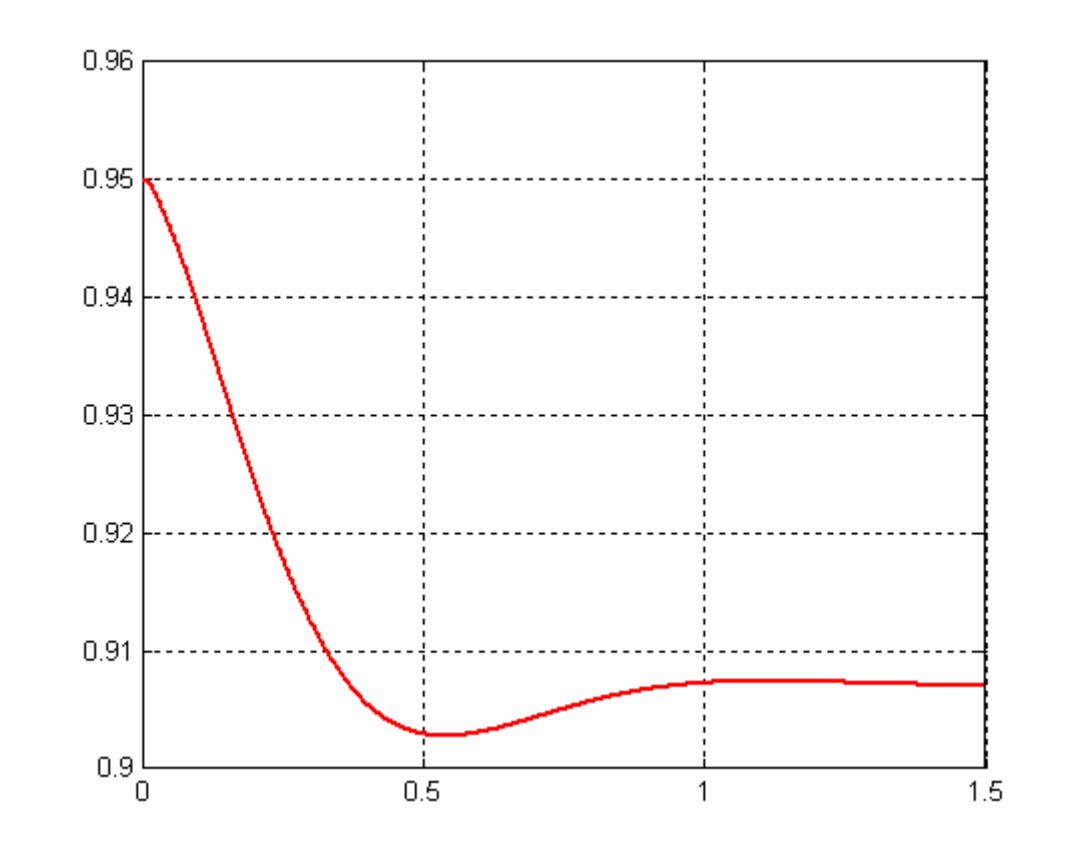

Figure 7.8: *Z*-coordinate of *Car Body*

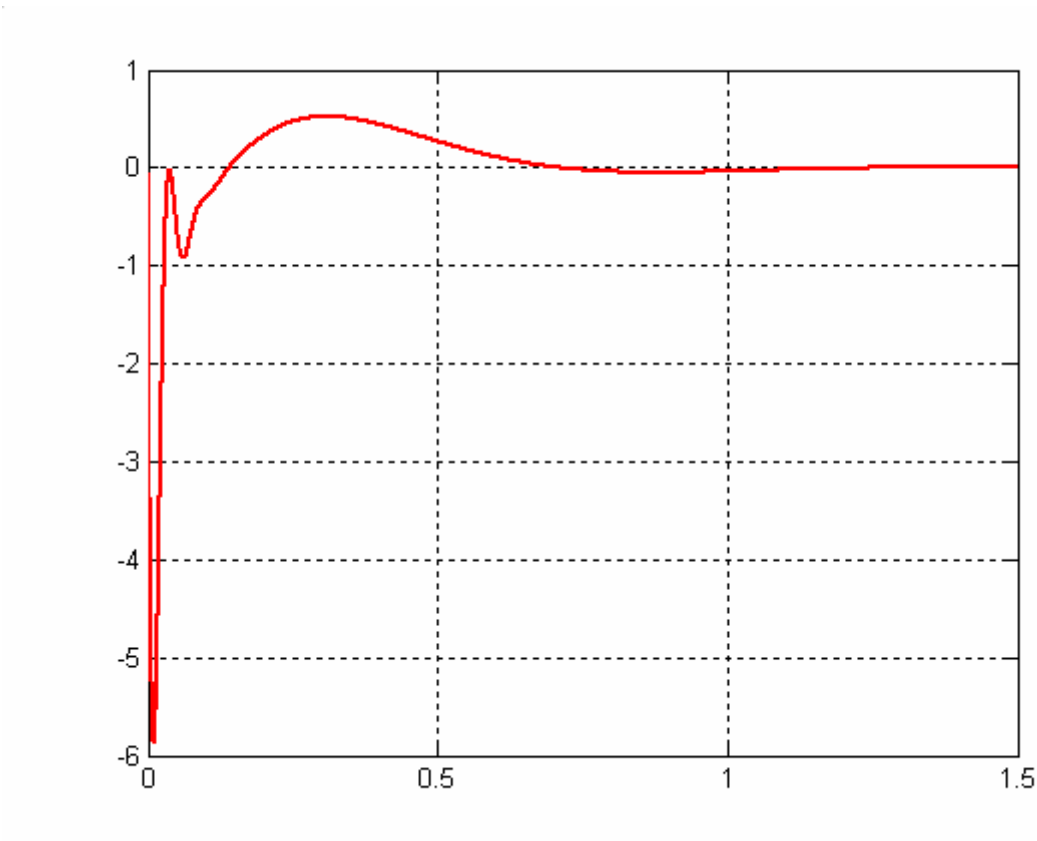

Figure 7.9: *Z*-acceleration of *Car Body*

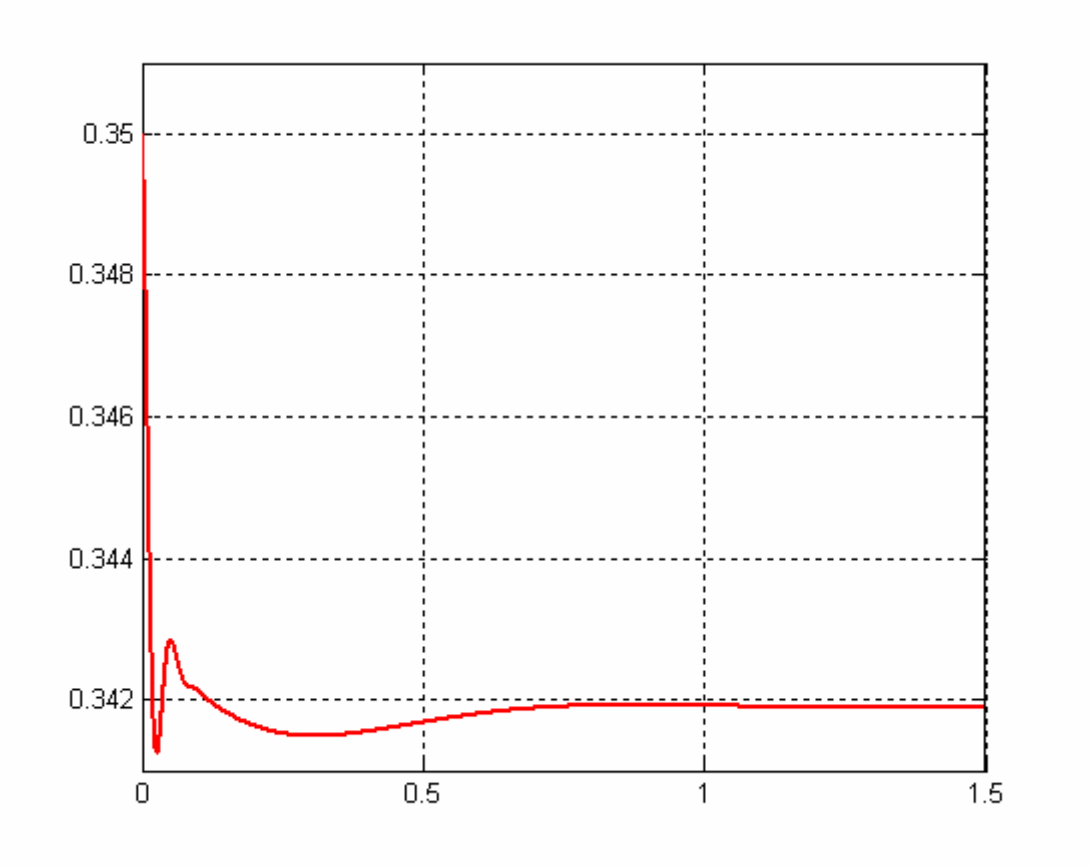

Figure 7.10: *Z*-coordinates of *Wheel Left*

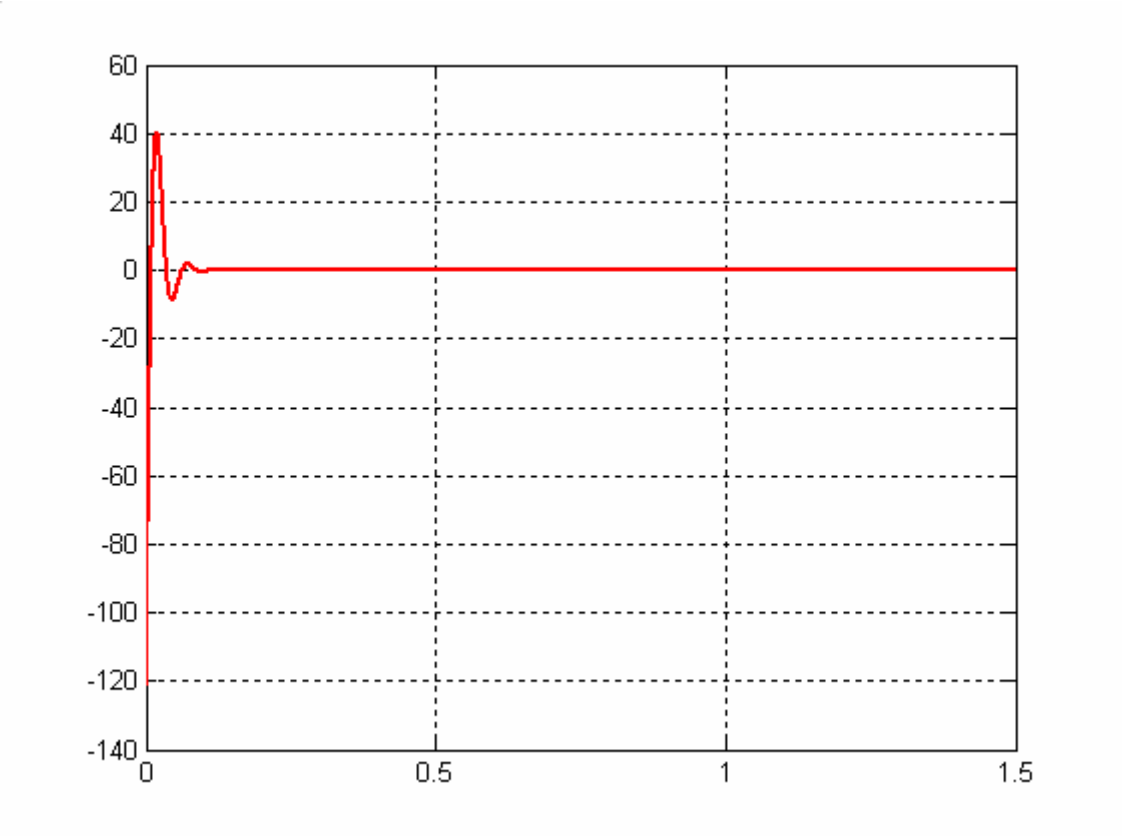

Figure 7.11: *Z*-acceleration of *Wheel Left*

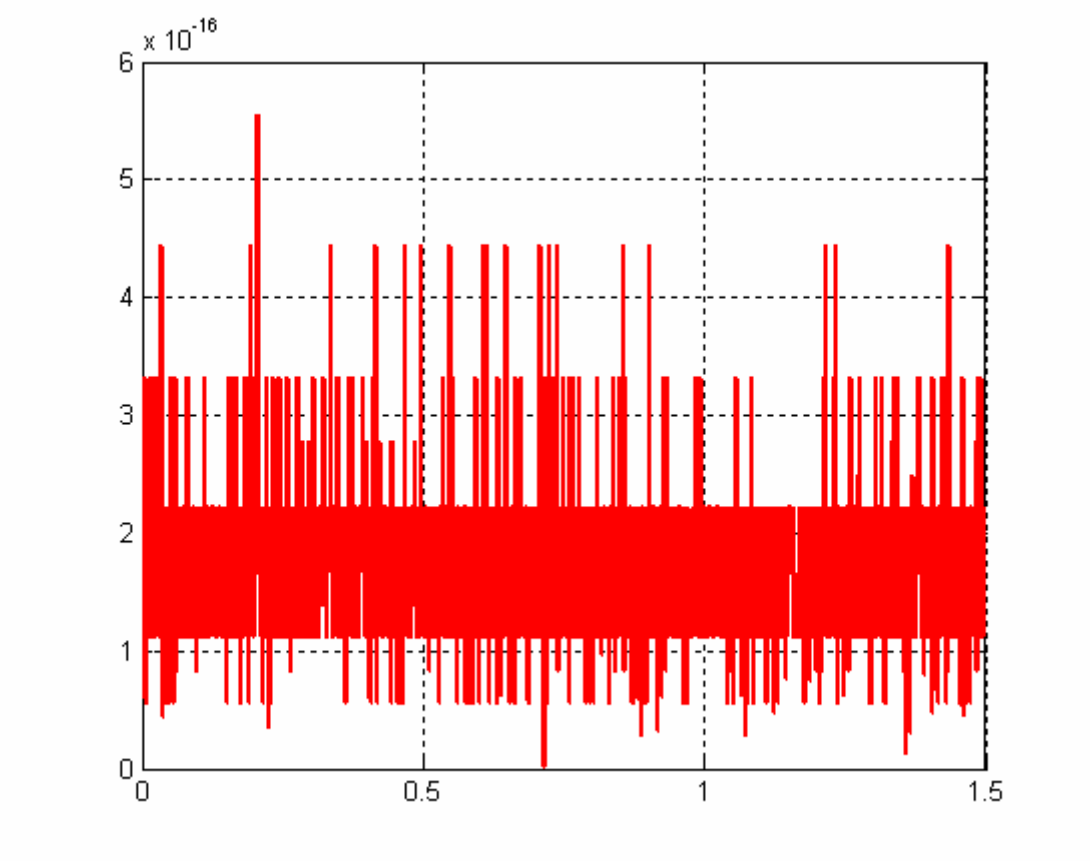

Figure 7.12: Drift of the model

For the validation of our simulations results we have built up the same car model in Simpack software, shown in Fig. 7.13. The simulation in Simpack was performed using Simpack's default integrator SODASRT, based on the DAE integrator DASSL.

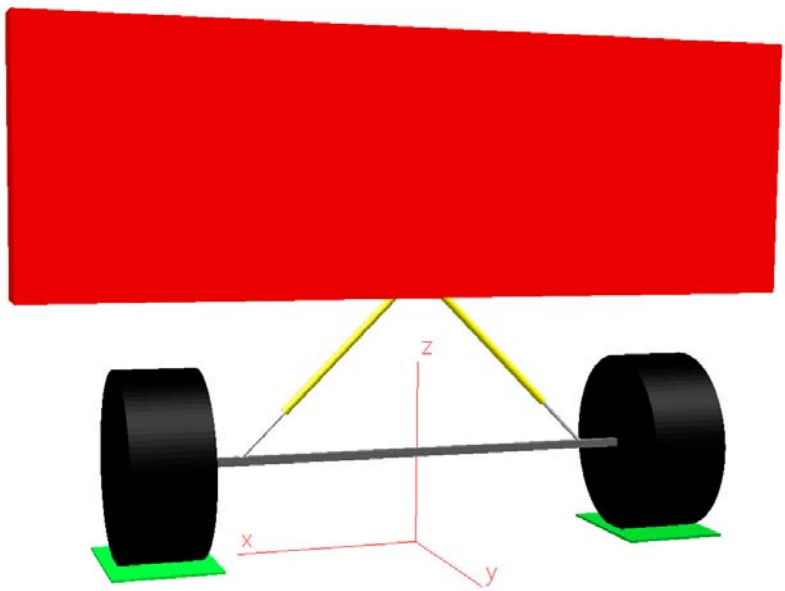

Figure 7.13: Simpack model

In the case of the simulation of closed-loop systems usually researchers perform the comparison of results on coordinate level (see, e.g. [HAI 96], [KUN 97]). In Fig. 7.14 – 7.15 are shown the absolute difference between z-coordinate of the car body and of the left wheel in our software and in Simpack.

Coordinate difference is limited by $1.4 \cdot 10^{-8}$, and stable. This result is comparable with other tests of DASSL integrator (e.g. the coordinate error of the simulation of a 2-D car truck in [KUN 97], coordinate error of the simulation of Andrew's squeezing mechanism in [HAI 96].

Therefore, the dynamics of the model was calculated correctly and our calculation algorithm is stable.

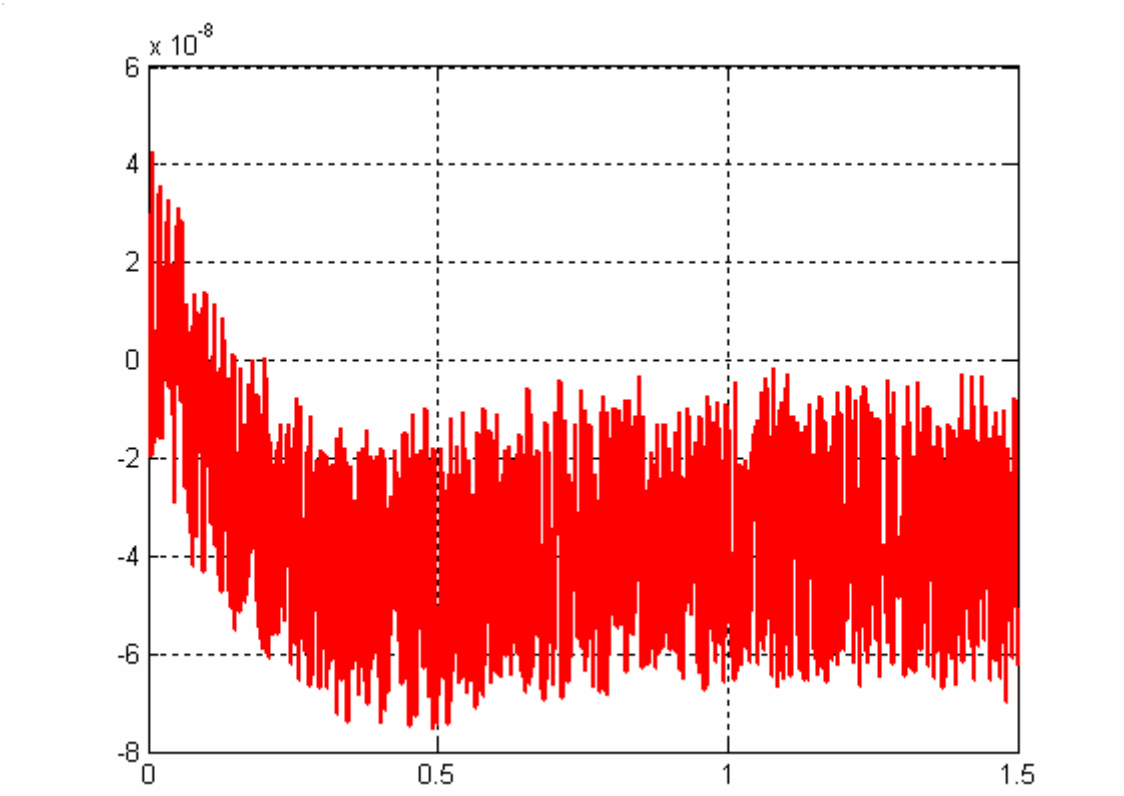

Figure 7.14: Absolute difference between z-coordinate of the car body in our software and in Simpack

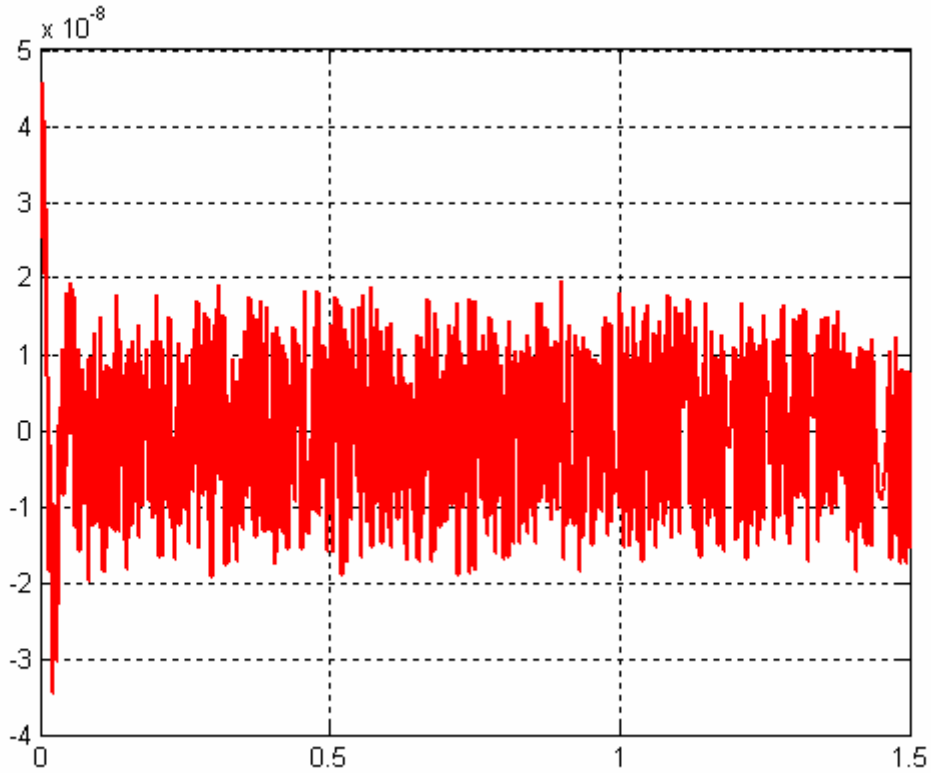

Figure 7.15: Absolute difference between z-coordinate of the left wheel in our software and in Simpack

8 Manipulator Example

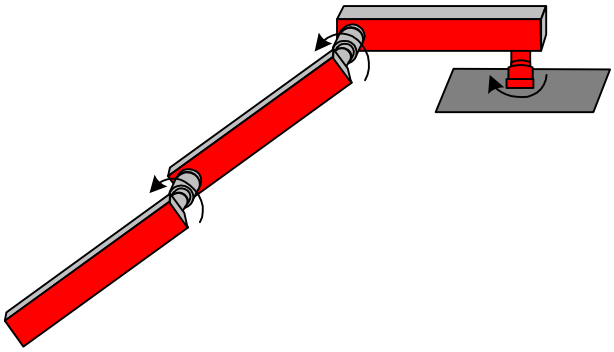

Figure 8.1: Three-links manipulator

As a second example we have performed a number of calculations for the problem of a three-links manipulator shown in Fig. 8.1.

Each link of the manipulator consists of a beam and a motor. All links are rigidly connected: each link's beam is fastened to the housing of the motor of the next link. The motor of the first link is rigidly connected to the ground.

The axis of rotation of the first motor is vertical and the axes of the two other motors are horizontal, that allows the manipulator to perform spatial movements.

The complete system consists of several subsystems: Motor, Link and Manipulator. This example illustrates the implementation of our method in a 3-D case. We perform the object-oriented simulation of the manipulator and compare the results of calculations using absolute and generalized coordinates with results of the simulation performed in Dymola software.

8.1 Motor subsystem

The Motor Subsystem shown in Fig. 8.2 describes the motor consisting of a housing (marked by red) and a rotor (marked by yellow) connected by a revolute joint. The motor torque T acts on the rotor in the forward direction and on the housing in the backward direction.

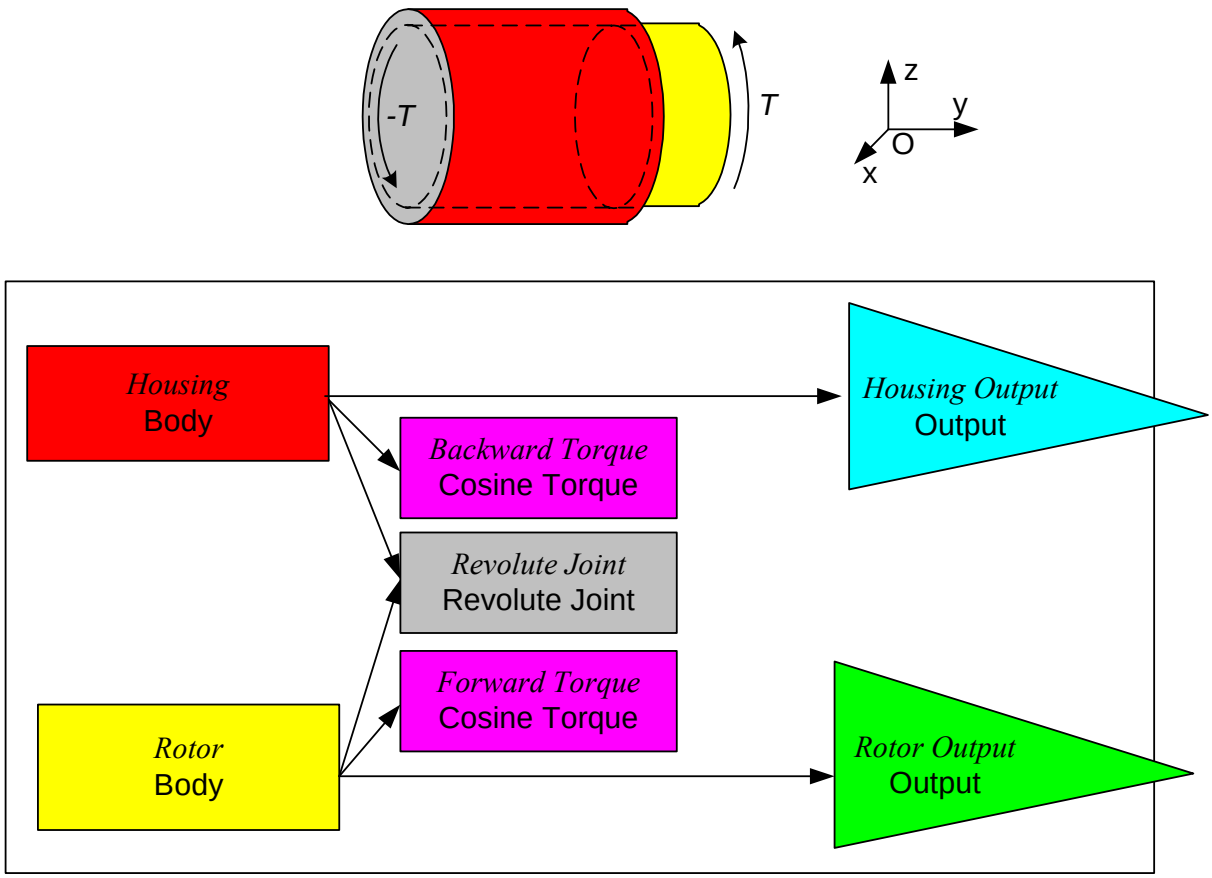

Figure 8.2: Motor

From the object-oriented point of view Motor is a Basic Subsystem consisting of two body objects (*Housing* and *Rotor*), *Revolute Joint*, two Cosine Torque objects (*Forward Torque* acting on the *Rotor* and *Backward Torque* acting on the *Housing*) and two Output objects (*Housing Output* and *Rotor Output*).

While the description of the subsystem we use the following parameters:

8.1.1 *Housing* parameters

1. m=2 - Mass,
2. h=0.3 - Height,
3. r=0.034 - Radius,

4. Δr=0.004 - Width of the walls.

8.1.2 *Rotor* parameters

1. m=7.7 - Mass,
2. h=0.35 - Height,
3. r=0.03 - Radius.

8.1.3 *Revolute Joint* parameters

1. $\boldsymbol{J}$={$Housing$, $Rotor$} - Array of connected bodies,
2. $\boldsymbol{B}$={$Housing$} - Array of basic bodies,
3. $\boldsymbol{K}$={$Rotor$} - Array of dependent bodies,
4. $\mathbf{r}_1$=(0 -0.15 0)^T - Distance from the housing's centre of mass to the joint,
5. $\mathbf{r}_2$=(0 -0.175 0)^T - Distance from the rotor's centre of mass to the joint,
6. $\mathbf{a}_1$=(0 1 0)^T - Axis of rotation.

8.1.4 *Forward Torque* parameters

1. $\boldsymbol{J}$={$Rotor$} - Array of bodies,
2. $\mathbf{a}_1$=(0 1 0)^T - Direction.

8.1.5 *Backward Torque* parameters

1. $\boldsymbol{J}$={$Housing$} - Array of bodies,
2. $\mathbf{a}_1$=(0 1 0)^T - Direction.

Remark 8.1 *There are three different motors in our manipulator model. That is why we set frequencies and amplitudes of their torques while the definition of Manipulator subsystem. The frequency of* $Backward\ Torque$ *is always equal to the frequency of a*

correspondent *Forward Torque* *and the amplitude of* *Backward Torque* *is always opposite to the amplitude of* *Forward Torque.*

8.2 Link Subsystem

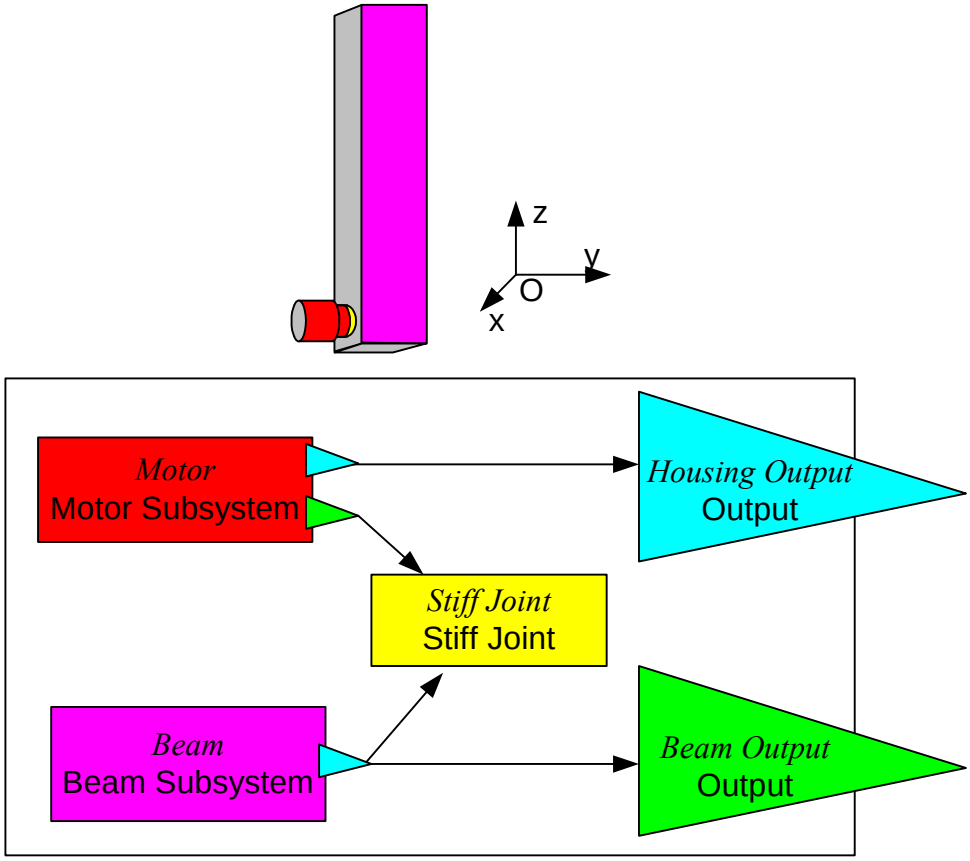

Figure 8.3: Link Subsystem

From the physical point of view the link shown in Fig. 8.3 is a mechanical subsystem consisting of a beam and a motor connected by a stiff joint.

From the object-oriented point of view Link Subsystem is a Derived Subsystem consisting of *Beam* (described in Chapter 7), *Motor*, *Stiff Joint*, and two Output objects (*Housing Output* and *Beam Output*).

While the description of the subsystem we use the following parameters:

8.2.1 *Beam* parameters

1. m=0.71 - Mass,
2. $\bar{\mathbf{J}} = diag(0.06, 0.06, 0.0002)$ - Moment of inertia.

8.2.2 *Stiff Joint* parameters

1. ***J***={*Motor.Rotor Output*, *Beam.Beam Output*} - Array of connected bodies,
2. ***B***={*Motor.Rotor Output*} - Array of basic bodies,
3. ***K***={*Beam.Beam Output*} - Array of dependent bodies,
4. $\mathbf{r}_1$=(0 0.175 0)^T - Distance from the rotor's centre of mass to the joint,
5. $\mathbf{r}_2$=(0 -0.02 -0.483)^T - Distance from the beam's centre of mass to the joint,
6. **s**=(1 0 0 0)^T - Euler parameters of the relative rotation.

8.2.3 Manipulator Subsystem

From the physical point of view the manipulator shown in Fig. 8.4 is a mechanical subsystem consisting of three links. All links are rigidly connected: each link’s beam is fastened to the housing of the motor of the next link.

From the object-oriented point of view Manipulator Subsystem is a Derived Subsystem consisting of three Link Subsystems, two Stiff Joints, and the Output object (*Housing Output*).

While the description of the Manipulator subsystem we use the following parameters:

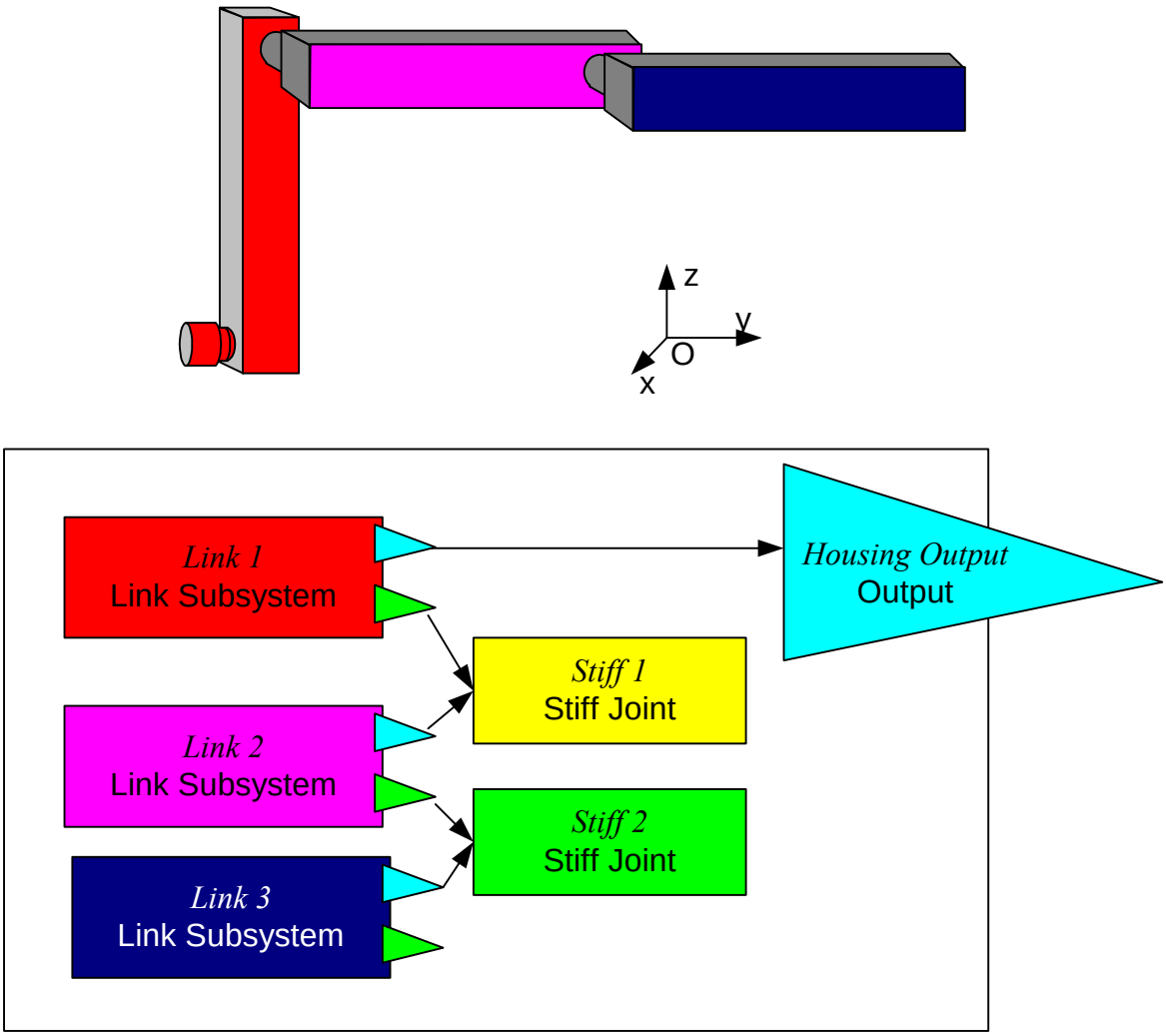

Figure 8.4: Manipulator Subsystem

8.2.4 *Stiff 1* parameters

1. ***J***={*Link 1.Beam Output*, *Link 2.Housing Output*} - Array of connected bodies,
2. ***B***={*Link 1.Beam Output* } - Array of basic bodies,
3. ***K***={*Link 2.Housing Output*} - Array of dependent bodies,
4. $\mathbf{r}_1=(0\ 0\ 0.5)^T$ - Distance from the beam's centre of mass to the joint,
5. $\mathbf{r}_2=(0\ -0.15\ 0)^T$ - Distance from the housing's centre of mass to the joint,

6. $\mathbf{s}=(0.5\ -0.5\ -0.5\ -0.5)^T$ - Euler parameters describing the relative rotation around the x-axis on the $\pi/2$ angle.

8.2.5 *Stiff 2* parameters

1. $\boldsymbol{J}=\{$*Link 2.Beam Output, Link 3.Housing Output*$\}$ - Array of connected bodies,
2. $\boldsymbol{B}=\{$*Link 2.Beam Output*$\}$ - Array of basic bodies,
3. $\boldsymbol{K}=\{$*Link 3.Housing Output*$\}$ - Array of dependent bodies,
4. $\mathbf{r}_1=(0\ 0.02\ 0.483)^T$ - Distance from the beam's centre of mass to the joint,
5. $\mathbf{r}_2=(0\ -0.15\ 0)^T$ - Distance from the housing's centre of mass to the joint,
6. $\mathbf{s}=(1\ 0\ 0\ 0)^T$ - Euler parameters of relative rotation.

8.2.6 *Link 1.Motor.Forward Torque* parameters

1. C=-600 - Amplitude,
2. k=2 - Frequency.

8.2.7 *Link 2.Motor.Forward Torque* parameters

1. C=300 - Amplitude,
2. k=2 - Frequency.

8.2.8 *Link 3.Motor.Forward Torque* parameters

1. C=-12 - Amplitude,
2. k=2 – Frequency.

8.3 Complete system

From the physical point of view the mechanical system shown in Fig. 8.5 consists of a manipulator and a ground connected by a stiff joint.

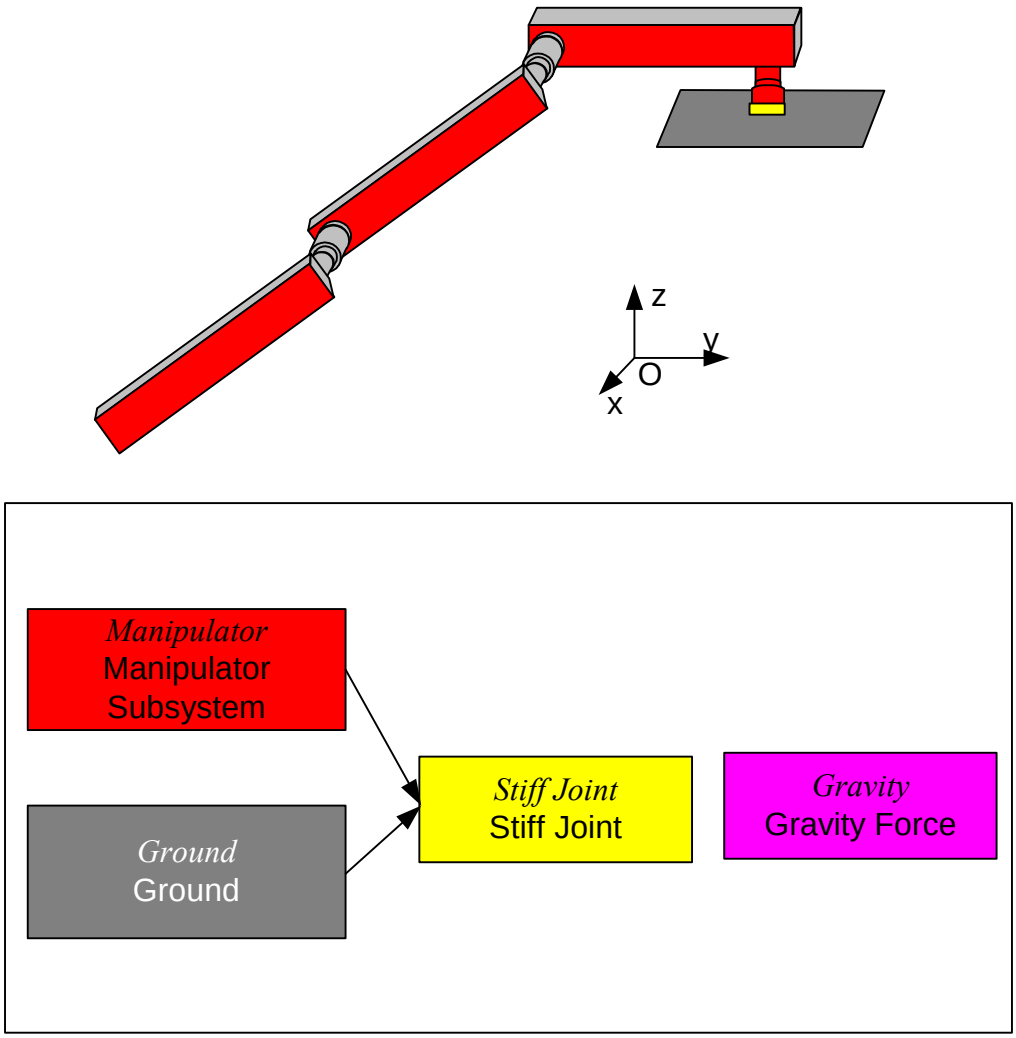

Figure 8.5: Complete system

From the object-oriented point of view the system is a Derived Subsystem consisting of *Manipulator*, *Ground*, *Stiff Joint* and *Gravity*.

While the description of the system we use the following parameters

8.3.1 *Stiff Joint* parameters

1. ***J***={*Ground, Manipulator.Housing Output*} - Array of connected bodies,
2. ***B***={*Ground*} - Array of basic bodies,
3. ***K***={*Manipulator.Housing Output*} - Array of dependent bodies,

4. $\mathbf{r}_1=(0\ 0\ 0)^T$ - Distance from the ground to the joint,

5. $\mathbf{r}_2=(0\ -0.15\ 0)^T$ - Distance from the housing's centre of mass to the joint,

6. $\mathbf{s}=\left(\frac{1}{\sqrt{2}}\quad\frac{1}{\sqrt{2}}\quad 0\quad 0\right)^T$ - Euler parameters describing the relative rotation around the *x*-axis on the -π/2 angle.

8.3.2 *Gravity* parameters

1. g=9.8 - Free fall acceleration,

2. $\mathbf{e}=(0\ 0\ -1)^T$ - Gravity direction.

8.4 Array of independent bodies and sequence of dependencies

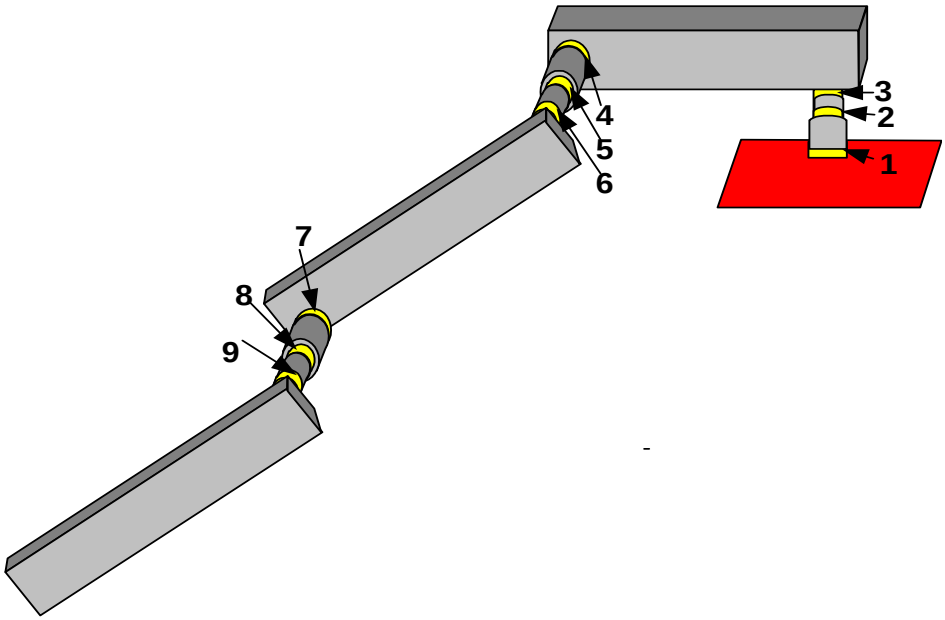

Fig. 8.6: Ground and Sequence of dependencies

The array of independent bodies $\boldsymbol{I}$ is null. The ground is marked in Fig. 8.6 by red.

The sequence of dependencies $\boldsymbol{C}$ consists of nine constraints (marked in Fig. 8.6 by yellow): $\boldsymbol{C}$={*Stiff Joint*, *Manipulator.Link 1.Motor.Revoulte Joint*, *Link 1.Stiff Joint, Manipulator.Stiff 1*, *Manipulator.Link 2.Motor.Revoulte Joint*, *Link 2.Stiff Joint*, *Manipulator.Stiff 2*, *Manipulator.Link 2.Motor.Revoulte Joint*, *Link 3.Stiff Joint*}. Constraint numbers in Fig. 8.6 are their order numbers in $\boldsymbol{C}$.

Using generalized coordinates while the simulation, we do not have drift problems because of the tree-structure of the manipulator.

8.5 Start values

The vector of generalized coordinates p consists of three elements:

1. The angle of rotation of *Manipulator.Link 1.Motor.Revoulte Joint*,
2. The angle of rotation of *Manipulator.Link 2.Motor.Revoulte Joint*,
3. The angle of rotation of *Manipulator.Link 3.Motor.Revoulte Joint*.

The start values of the generalized coordinates and velocities are:

$$\mathbf{p}|_{t=0} = \begin{pmatrix} 0 & 2.5 & 0 \end{pmatrix}^T$$
$$\mathbf{w}|_{t=0} = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T$$

8.6 Simulation data

We perform the simulation of the model using the method described in Chapter 2. We choose the time interval to be [0, 5]. Simulation was performed with predictor-corrector method of Adams-Moulton with the fixed time step equal to $10^{-4}s$.

In Fig. 8.7-8.9 are shown the changes of generalized coordinates, generalized velocities and generalized accelerations.

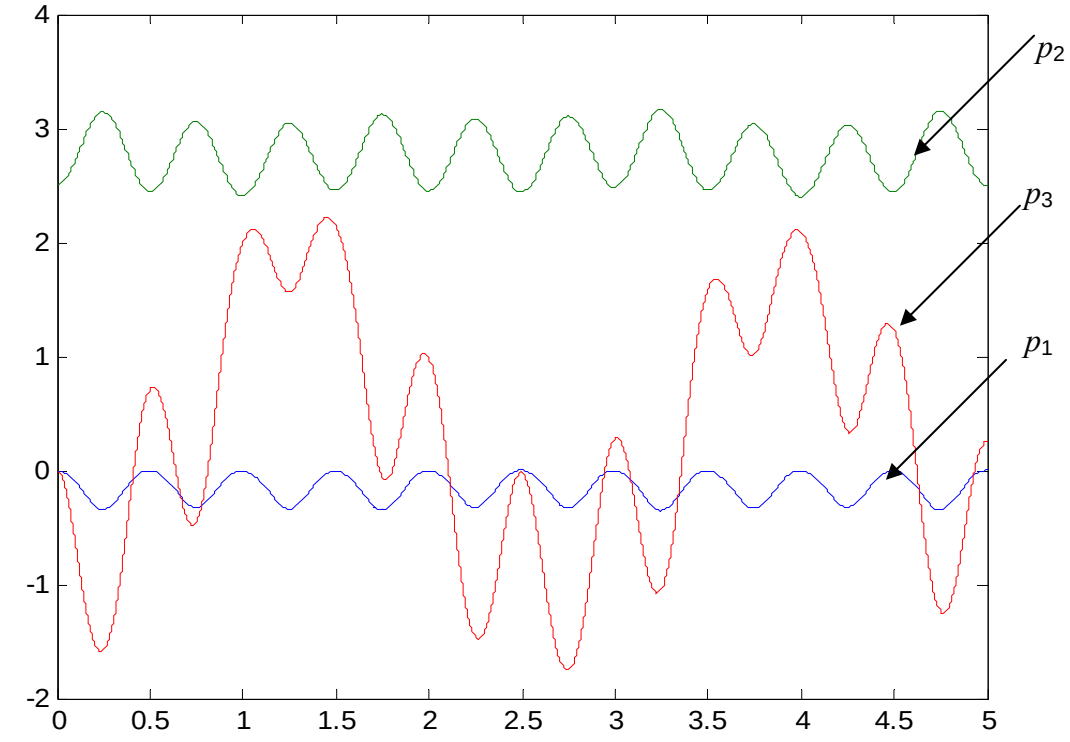

Figure 8.7: Generalized coordinates

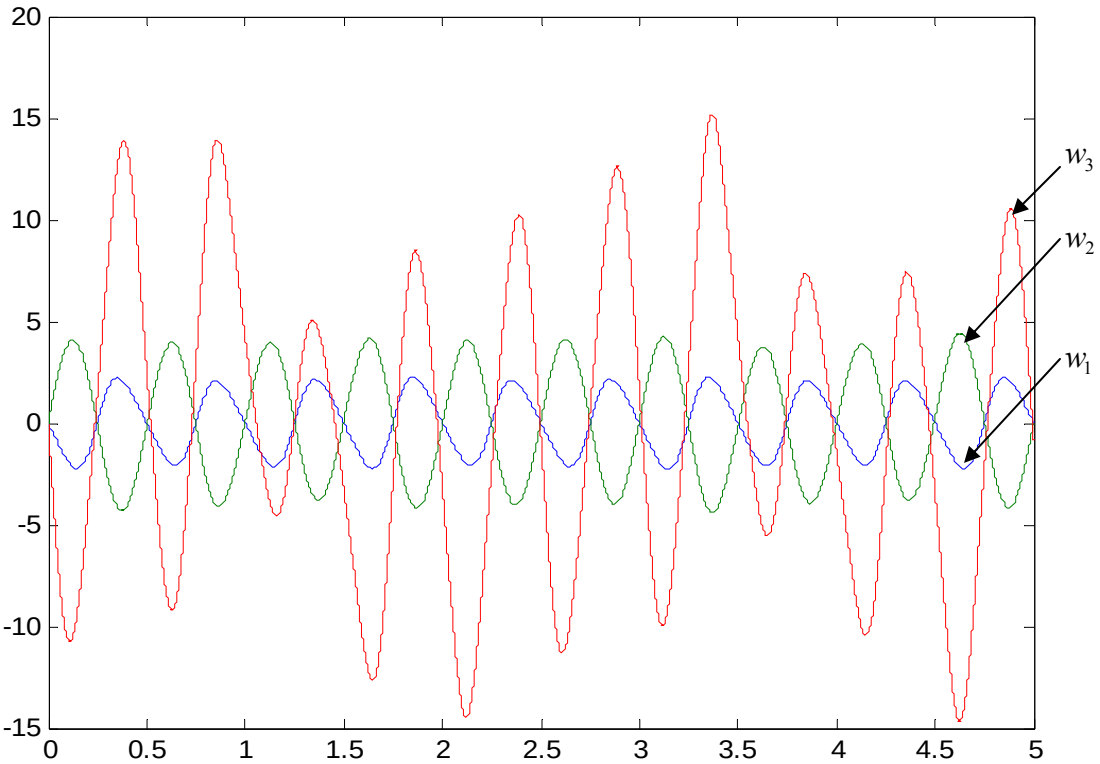

Figure 8.8: Generalized velocities

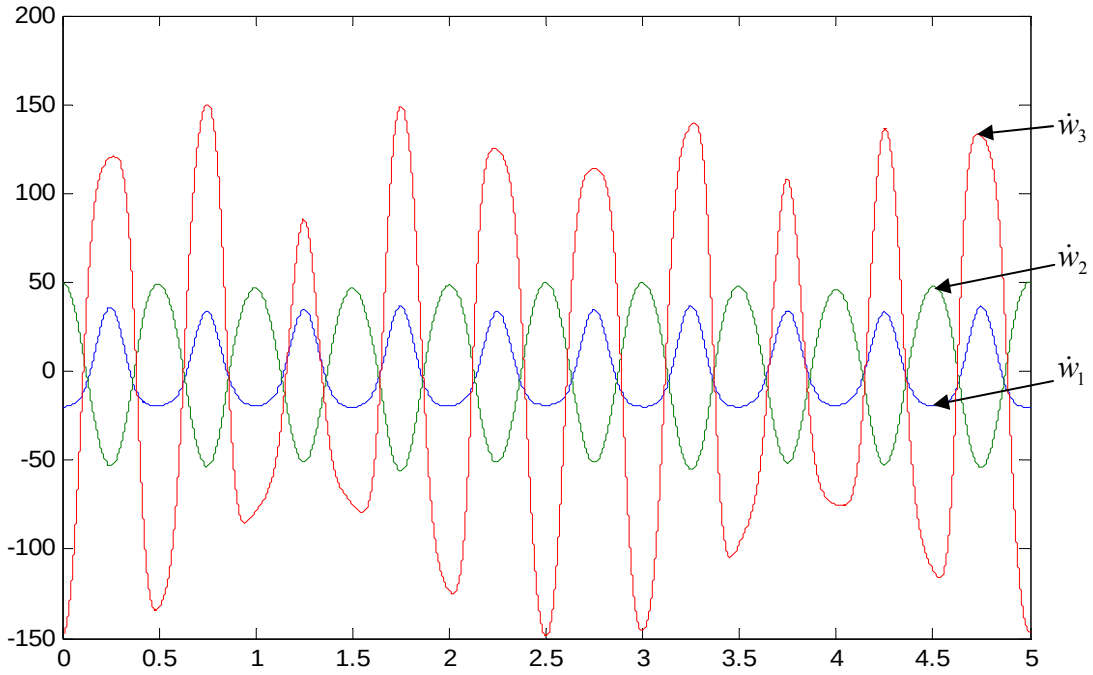

Figure 8.9: Generalized accelerations

For the validation of our simulations results we have built up the same manipulator model in Dymola software, shown in Fig. 8.10.

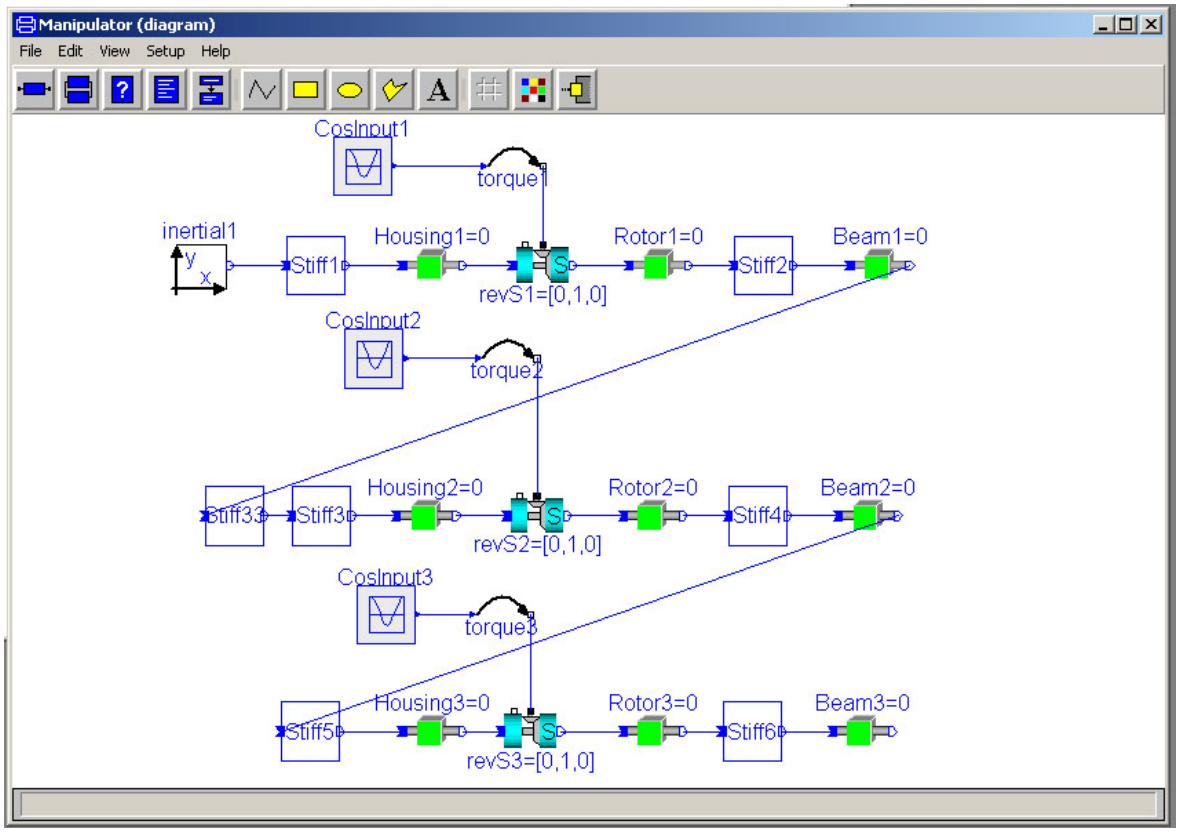

Figure 8.10: Dymola manipulator model

Obviously, it is most sensible to compare our simulation results on the accelerations level because of their high variability. In Fig. 8.11 is shown the maximum absolute difference $\Delta\dot{\mathbf{w}}(t)$ between generalized accelerations in our software and in Dymola, that are calculated using the formula:

$$\Delta\dot{\mathbf{w}}(t) = \max_{i=1..3} \left| \dot{\mathbf{w}}_i^D(t) - \dot{\mathbf{w}}_i(t) \right|$$

where

$\dot{\mathbf{w}}_i^D(t)$ is the i-th generalized acceleration calculated in Dymola,

$\dot{\mathbf{w}}_i(t)$ is the i-th generalized acceleration calculated in our software.

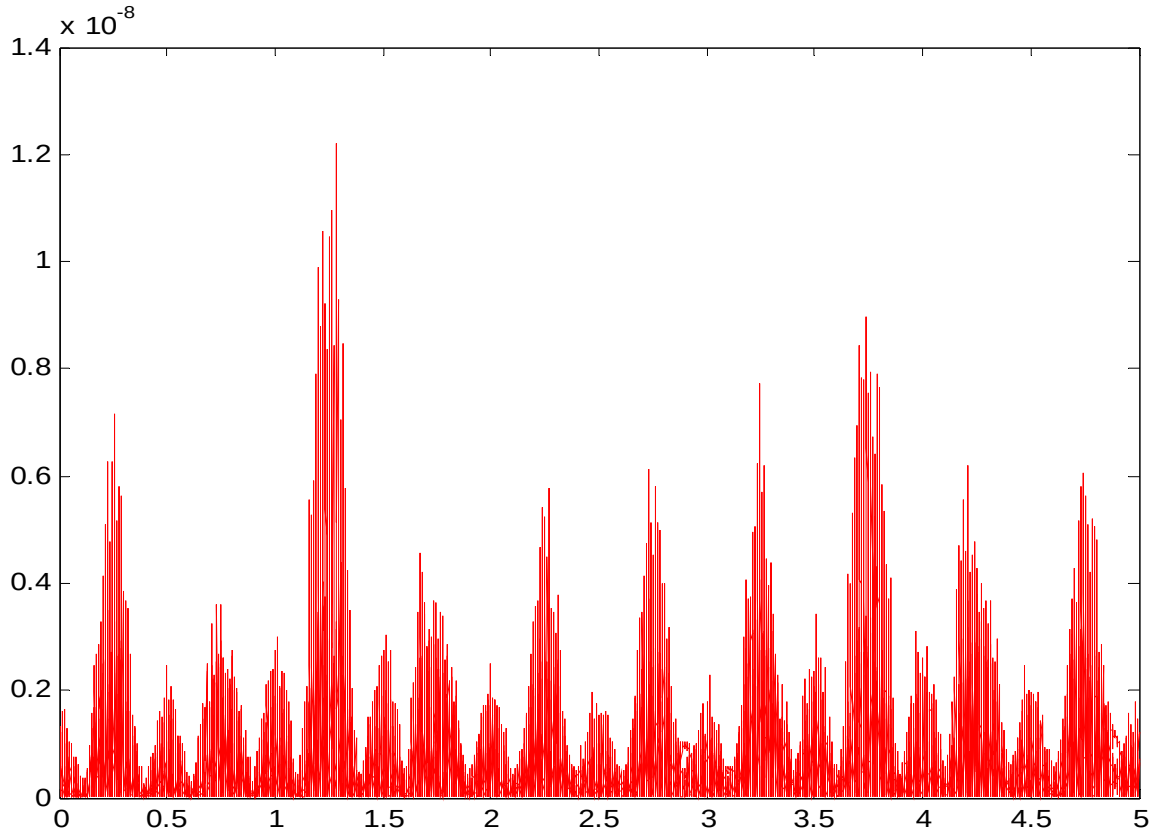

Figure 8.11: Difference between the generalized accelerations in our software and in Dymola

The comparison of accelerations shows that the dynamics of the manipulator model was calculated correctly and our calculation algorithm is stable. Peaks in Fig. 8.11 are limited by $1.4 \cdot 10^{-8}$ and correspond to local extremums of accelerations.

If we perform the simulation of the model using absolute coordinates without stabilization, the error in constraints' equations grows with time. In Fig. 8.12 is shown the drift of the unstabilized model.

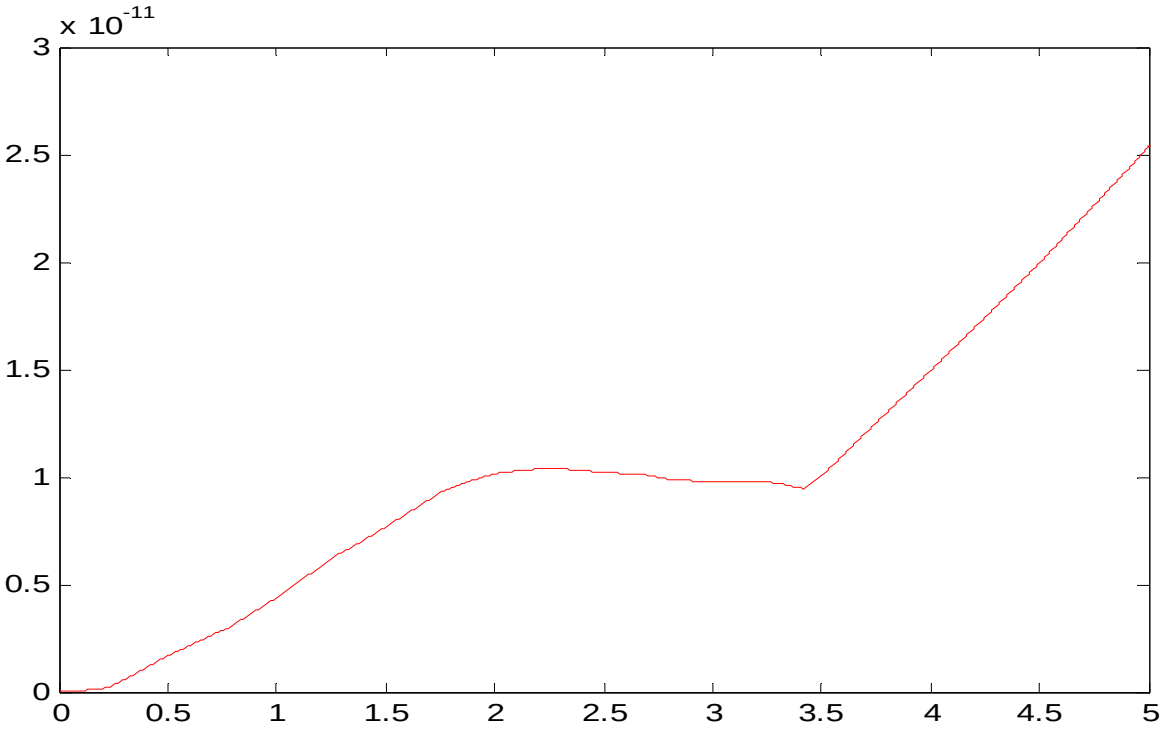

Figure 8.12: Drift of the unstabilized model

Conversely, the drift of the stabilized model shown in Fig. 8.13 is limited for a long period of time and has the order of floating-point precision.

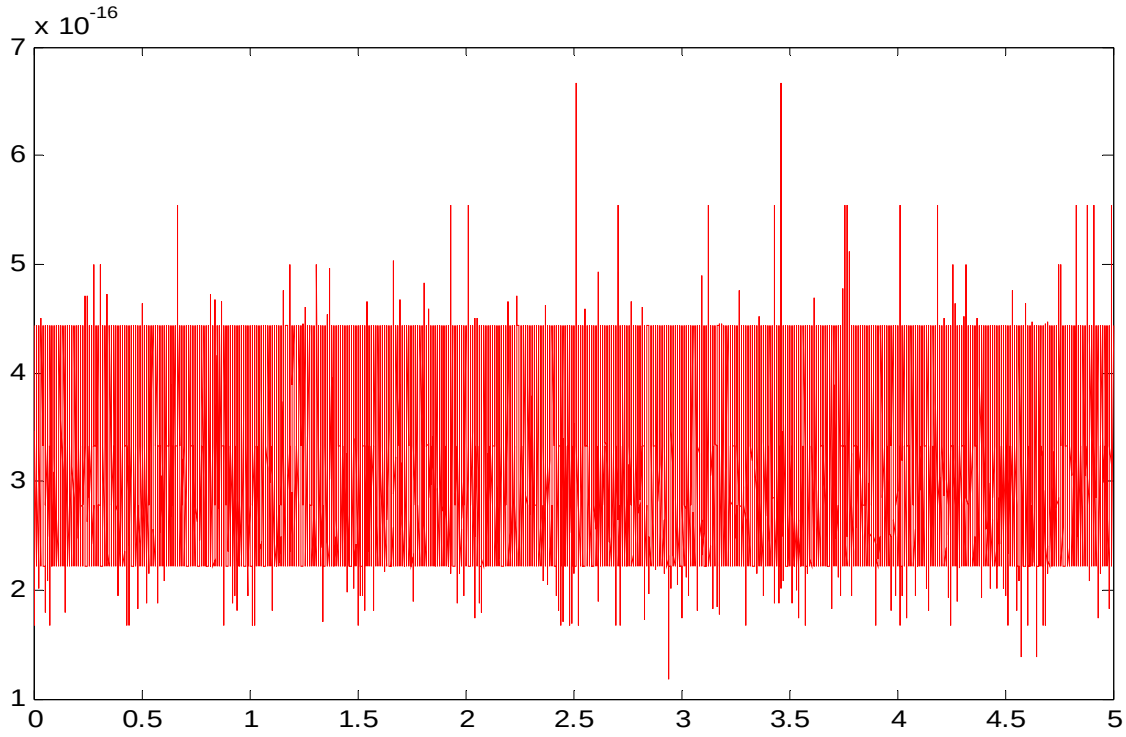

Figure 8.13: Drift of the stabilized model

For the validation of simulations results of the stabilized results we compared them with Dimola's results. It is the most sensible to compare accelerations of *Link 3* because of their highest variability. In Fig. 8.14 is shown the maximal absolute difference between accelerations of *Link 3* in our stabilized model and in Dymola.

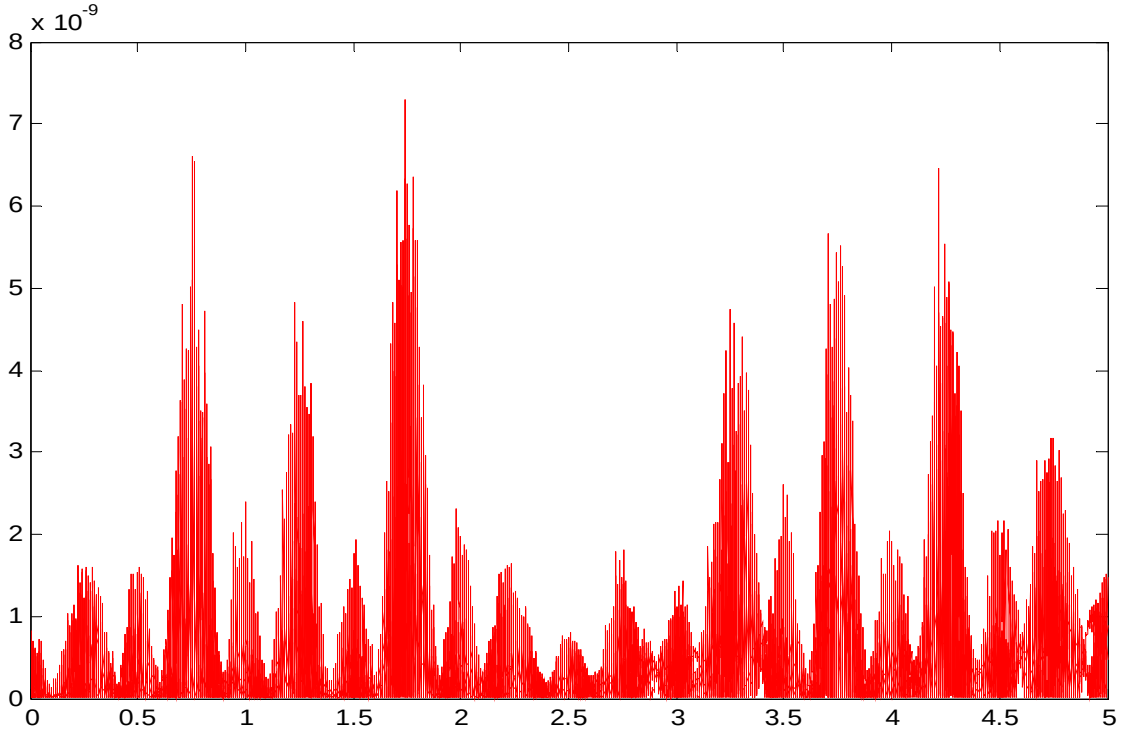

Figure 8.14: Difference between the accelerations of *Link 3* in the stabilized model and in the Dymola model

Therefore, the stabilization works correctly though the total number of absolute coordinates is 64 and the total number of equations of constraints is 75.

We get that the manipulator’s model can be correctly simulated in two ways: the first is the using of generalized coordinates, the second is the performing post-stabilization using absolute coordinates. The using of generalized coordinates has the less computation complexity that is very important in the simulation of multibodies.

Finally, the simulation results show that our tool does not have limits on the structure of the simulating model and can be implemented for the simulation of complex 3-D models.

9 Conclusion

9.1 Results

Our goal is to develop a method for component oriented modelling and simulation of constrained multibody dynamics.

In this thesis we start from the comparison of forward dynamic methods that can be used as bases of simulation tools. We determine the most important appreciated characteristics of methods: stability, numerical efficiency and practical usability. Comparing the different stabilization technique, we show the advantage of the post-stabilization technique [AHR 95]. In Chapter 1 are observed also a few methods of distributed forward dynamic simulation. Nowadays, the fastest available method, divide-and-conquer algorithm, has a large number of drawbacks and limits. That is why we develop the new distributed object-oriented method that is more stable and convenient in practical use.

An implementation of a method is not trivial and requires great effort. In Chapter 1 we determine the main characteristics of simulation tools: flexibility, usability and interaction with other tools. We precisely describe the object-oriented modelling paradigm that we use as the basis of our tool.

In Chapter 2 we observe some important theoretical problems of the development of our method i.e. the choice between generalized and absolute coordinates, choice of absolute coordinates etc. We show that the method is stable, distributable and does not have limits on the structure of a simulating system.

In Chapter 3 we appreciate the several computation complexities: stabilization, a simulation of a basic subsystem and simulation of a derived subsystem. Summarizing them, we obtain the global $O(n \cdot D^3 + t^2 \cdot s)$ complexity of the method, where n is the total number of bodies, D is the upper limit of constraints in a subsystem, t is the number of closed loops and s is the total number of bodies in loops. Thus, the numerical efficiency of our method is comparable with fastest available algorithms.

In Chapter 4 we show the implementation background of the method. The software is based on a strictly capsulated block-module concept. In this context it means that the mechanical structure will be represented by separate objects which interact with each other via predefined interfaces. Then we precisely consider the advantages of such an approach and its profits of calculation of accelerations using the same hierarchy disassembly as it was performed while the model's construction.

In Chapter 5 we start the consideration of the object-oriented implementation of our method. We show eight basic objects that are used in our algorithm: Timer, Ground, Body, Body Output, Generalized Force, Constraint, Basic subsystem, Derived subsystem. In Chapter 6 we show the child objects that describe the different types of constraints and forces: Revolute Joint, Ball Joint, Gravity Force etc.

Using the Visual-Basic implementation of our method, in Chapter 7 and Chapter 8 we simulate two models: a car system and a spatial manipulator. Both models are performed using object-oriented approach, with several levels of hierarchy.

The simulation data shows that the algorithm is stable and the model's drift is constant and has the order of the computation accuracy in the cases of closed-loop and tree structure.

For the validation of our simulation results we have built up the same models in Dymola and Simpack software. The comparison shows that the dynamics of the models was calculated correctly.

Thus, we obtain the simulation proof that our tool could be implemented for the simulation of large constrained multibody systems.

9.2 Discussion of future work

9.2.1 Integration with CAD tools

It seems to be inconvenient to create a new graphical model editor like Dymola Editor or Simulink because of the high cost of the development and the existence of other editors. Since most of 3D models are created inside CAD tools, the much more effective way is to integrate our tool with CAD tools. In this case a design engineer

specifies geometric and material data of simulation model inside CAD tool and then translates it into our simulation tool. This approach minimise the model's development cost and training of design engineers.

9.2.2 Simulations and analysis of systems with variable structures

Many industrial systems such as robots are subjected to a change in their kinematic structure during the simulation. Backslash and coulomb friction are possible sources of these mechanical structure changes. In this case the structure and the number of equations of constraints changes and discontinuities on acceleration level will occur. Simulations and dynamical analysis of multibody systems with variable kinematic structures are needed. The method presented here can be extended straightforward to deal with this variable structures.

9.2.3 Distributed simulation

Nowadays, we consequently perform the transformation from generalized coordinates and velocities to absolute coordinates and velocities. The next step is to develop the method of distribution of this calculation using the existing models hierarchy.

The sophisticated problem is the implementation of the distribution. Classical methods of distributed simulation works on computers with many processors and low communication cost. But the much more common situation is a network consisting of several computers. In this case we should minimise the communication costs. The optimal way is to start on each computer an independent procedure that translates and simulates an individual subsystem. The development of the interaction of computers during the simulation is a challenge.

Appendix A

Quaternions algebra

A quaternion is a collection of four real parameters, of which the first is considered as a scalar and the other three as a vector in three-dimensional space. In addition, the following operations are defined. If $\boldsymbol{\theta} = \begin{pmatrix} e_0 & \mathbf{e}^T \end{pmatrix}^T = \begin{pmatrix} e_0 & e_1 & e_2 & e_3 \end{pmatrix}^T$ and $\boldsymbol{\varsigma} = \begin{pmatrix} c_0 & \mathbf{c}^T \end{pmatrix}^T = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \end{pmatrix}^T$ are two quaternions, their sum is defined as

$$\boldsymbol{\theta} + \boldsymbol{\varsigma} = \begin{pmatrix} e_0 + c_0 & \mathbf{e}^T + \mathbf{c}^T \end{pmatrix}^T \tag{A. 1}$$

and their product (non-commutative) as

$$\boldsymbol{\theta} \circ \boldsymbol{\varsigma} = \begin{pmatrix} e_0 c_0 - \mathbf{e} \cdot \mathbf{c} \\ e_0 \mathbf{c} + c_0 \mathbf{e} + \mathbf{e} \times \mathbf{c} \end{pmatrix} \tag{A. 2}$$

Thus, differentiating the product, we obtain

$$(\boldsymbol{\theta} \circ \boldsymbol{\varsigma})' = \boldsymbol{\theta}' \circ \boldsymbol{\varsigma} + \boldsymbol{\theta} \circ \boldsymbol{\varsigma}' \tag{A. 3}$$

The quaternion $\boldsymbol{\theta}$ is identified as the set of Euler parameters for the description of finite rotation. According to Euler's theorem of finite rotation, a rotation in space can always be described by a rotation along a certain axis over a certain angle. With the unit vector $\mathbf{a}_\mu$ representing the axis and the angle of rotation μ, right-handed positive, the Euler parameters $\boldsymbol{\theta}$ can be interpreted as

$$e_0 = \cos(\mu/2) \quad \mathbf{e} = \sin(\mu/2)\mathbf{a}_\mu \tag{A. 4}$$

Since the definition, it follows that

$$e_0^2 + e_1^2 + e_2^2 + e_3^2 = 1$$

The rotational matrix $\mathbf{A}(\boldsymbol{\theta})$ is equal:

$$\mathbf{A}(\boldsymbol{\theta}) = \begin{pmatrix} e_0^2 + e_1^2 - e_2^2 - e_3^2 & 2(e_1e_2 - e_0e_3) & 2(e_1e_3 + e_0e_2) \\ 2(e_2e_1 + e_0e_3) & e_0^2 - e_1^2 + e_2^2 - e_3^2 & 2(e_2e_3 - e_0e_1) \\ 2(e_3e_1 - e_0e_2) & 2(e_3e_2 + e_0e_1) & e_0^2 - e_1^2 - e_2^2 + e_3^2 \end{pmatrix} \tag{A. 5}$$

The rotation matrix $\mathbf{A}$ of two consecutive rotations $\boldsymbol{\theta}$ and $\boldsymbol{\varsigma}$ is equal:

$$\mathbf{A} = \mathbf{A}(\boldsymbol{\theta}) \cdot \mathbf{A}(\boldsymbol{\varsigma}) = \mathbf{A}(\boldsymbol{\theta} \circ \boldsymbol{\varsigma}) \tag{A. 6}$$

A simple relationship exists between the components of the global angular velocity vector $\boldsymbol{\Omega}$ and time derivatives of Euler parameters $\dot{\boldsymbol{\theta}} = \begin{pmatrix} \dot{e}_0 & \dot{e}_1 & \dot{e}_2 & \dot{e}_3 \end{pmatrix}^T$:

$$\dot{\boldsymbol{\theta}} = \frac{1}{2}\mathbf{E}^T\boldsymbol{\Omega},$$

where $\mathbf{E}$ is a semi-transformation matrix that depends linearly on Euler parameters:

$$E = \begin{pmatrix} -e_1 & e_0 & -e_3 & e_2 \\ -e_2 & e_3 & e_0 & -e_1 \\ -e_3 & -e_2 & e_1 & e_0 \end{pmatrix} \tag{A. 7}$$

Also exists the backward dependency:

$$\Omega = 2\mathbf{E}\dot{\boldsymbol{\theta}} \tag{A. 8}$$

Bibliography

[ALI 92] T. Alishenas. *Zur numerischen Behandlung, Stabilisierung durch Projektion und Modellierung mechanischer Systeme mit Nebenbedinungen und Invarianten*, PhD thesis, Königliche Technische Hochschule Stokholm (1992)

[AHR 93] U. Ascher and L. Petzold, *Stability of Computational Methods for Constrained Dynamic Systems*, SIAM J. SISC 14, pp. 95-120 (1993)

[AHR 95] U. Ascher, H. Chin, L. Petzold and S. Reich, *Stabilization of constrained mechanical systems with DAEs and invariant manifolds*, The Journal of Mechanics of Structures and Machines, 23(2), pp. 135-157(1995)

[AHR 98] Ascher, U. and Petzold, L., *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*, SIAM, Philadelphia (1998)

[AND 00] Anderson, K. and Duan, S., *Highly parallelizable low-order dynamics simulation algorithm for multi-rigid-body systems*, AIAA Journal on Guidance, Control and Dynamics 23, no. 2, pp. 355-364 (2000)

[ARM 79] Armstrong, W., *Recursive solution to the equations of motions of an n-link manipulator*, Proc. 5th World Congress on Theory of Machines and Mechanisms, Montreal, pp. 1343-1346 (1979)

[BAE 87] Bae, D. and Haug, E., *A recursive formulation for constrained mechanical system dynamics: part 2. Closed loop systems*, Mech. Struct. Mach. 15, no. 4, pp. 481-506 (1987)

[BAE 99] Bae, D. and Han, J., *A generalized recursive formulation for constrained mechanical system dynamics*, Mech. Struct. Mach. 27, no. 3, pp. 293-315 (1999)

[BAU 72] J. Baumgarte, *Stabilization of constraints and integrals of motion in dynamical systems*, Comp. Math. Appl. Mech. Eng. 1, pp. 1-16 (1972).

[BIR 87] Birta, L. and Abou-Rabia, A., *Parallel block predictor-corrector methods of ODE's*, IEE Trans. Computers C-36, pp. 299-311 (1987)

[BRA 86] Brandl, H., Johanni, R., and Otter, M., *A very efficient algorithm for the simulation of robots and similar multibody systems without inversion of the mass matrix*, Proc. IFAC/IFIP/IMACS International Symposium on Theory of Robots, Vienna (1986)

[BRE 89] K. Brenan, S. Campbell and L. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, North-Holland (1989)

[CEL 95] Cellier, F.E., H. Elmqvist, and M. Otter, *Modeling from Physical Principles*, The Control Handbook (W.S. Levine, ed.), CRC Press, Boca Raton, FL (1995)

[CLI 03] M. B. Cline and D. K. Pai *Post-Stabilization for Rigid Body Simulation with Contact and Constraints* Proc IEEE Intl. Conf. on Robotics and Autom., 2003 (2003)

[CHA 90] Chaudhry, V. and Aggarwal, J., *Parallel Algorithms for Machine Intelligence and Vision*, Chap. Parallelism in Computer Vision: A Review, Springer-Verlag, New York (1990)

[CHI 95] H. Chin, *Stabilization methods for simulation of constrained multibody dynamics,* PhD thesis, Institute of Applied Mathematics, University of British Columbia (1995)

[EIC 93] Eichberger, A., C. Führer, et al., *The Benefits of Parallel Multibody Simulation and its Application to Vehicle Dynamics. Advanced Multibody System Dynamics - Simulation and Software Tools*, W. Schiehlen. Dordrecht, Kluwer Academic Publishers, pp. 107-126 (1993)

[ELM 78] Elmqvist, H., *A Structured Model Language for Large Continuous Systems*, Ph. D. Dissertation, Report CODEN: LUTFD2/(TFRT-1015), Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden (1978)

[ELM 01] Elmqvist, H., Mattsson, S. E., Otter, M., *Object-Oriented and Hybrid Modeling in Modelica*, Journal Européen des systèmes automatisés, 35,1, pp. 1 – 10 (2001)

[FEA 83] Featherstone, R., *The calculation of robot dynamics using articulated-body inertias*, Int. Journal of Robotics Research 2, no. 1, pp. 13-30, (1983)

[FEA 87] Featherstone, R., *Robot Dynamics Algorithms*, Kluwer Academic Publishers, Boston/Dordrecht/Lancaster (1987)

[FEA 99] Featherstone, R. and Fijani, A., *A technique for analyzing constrained rigid-body systems, and its application of the constraint force algorithm*, IEEE Trans. Robotics and Automation 15, no. 6, pp. 1140-1144 (1999)

[FEA 99a] Featherstone, R., *A divide-and-conquer articulated-body algorithm for parallel O*(*log*(*n*)) *calculation of rigid-body dynamics, part 2: trees, loops and accuracy*, Int. Journal of Robotics Research 18, no. 9, pp. 876-892 (1999)

[FUJ 92] Fijani, A. and Bejczy, A., *Parallel Computation Systems for Robotics: Algorithms and Architectures*, World Scientific, River Edge, NJ (1992)

[FUJ 95] Fijani, A., Sharf, L, and D'Eleuterio, C. M., *Parallel O*(*logN*) *algorithms for computation of manipulator forward dynamics*, IEEE Trans. Robotics and Automation 11, no. 3, pp. 389-400 (1995)

[GAR 94] Garcia de Jalon, J. and Bayo, E., *Kinematic and Dynamic Simulation of Multi-body Systems: The Real-Time Challenge*, Springer-Verlag, New York (1994)

[GEA 81] C.W. Gear, H. H. Hsu and L. Petzold, *Differential-algebraic equations revisited*, Proc. ODE Meeting, Oberwolfach, West Germany, (1981)

[GEA 85] C.W. Gear, G. Gupta and B. Leimkuhler, *Automatic integration of the Euler-Lagrange equations with constraints*, J. Comput. Appl. Math. 12, pp. 77-90 (1985)

[GOL 96] G.H. Golub and C.F. van Loan, *Matix computations. Second Edition,* John Hopkins University Press, Baltimore (1996)

[HAI 96] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations. II. Stiff and Differential-Algebraic Problems.* Springer–Verlag, Berlin Heidelberg New York, 2nd edition (1996)

[HAU 90] E.J. Haug, *Computer-Aided Kinematics and Dynamics of Mechanical System* Vol.1:Basic Methods, Allyn and Bacon (1989)

[HEN 97] D. Henrich, T. Honiger, *Parallel processing approaches in robotics*, Proc. IEEE International Symposium on Industrial Electronics, Guimaraes, Portugal, pp. 702-707 (1997)

[JAI 91] A. Jain, *Unified formulation of dynamics for serial rigid multibody systems*, Journal of Guidance, Control, and Dynamics, 14, pp. 531-542 (1991)

[KAS 95] Kasper, R. and Koch W. *Object-Oriented Behavioural Modelling of Mechatronic Systems,* Proceedings of the Third Conference on Mechatronics and Robotics, Stuttgart: Teubner (1995)

[KAS 97] Kasper, R.; Koch, W.; Kayser, A.; Wolf, A. *Integrated design environment for mechatronic components and systems of automotive industry*, VDI Bericht 1374, pp. 451-465 (1997)

[KAS 99] Kasper, R.; Koch, W.: *An Innovative Mechatronic Design Environment Based on COM Technology and ActiveX*, Proceedings of 3rd International Heinz Nixdorf Symposium, Paderborn (1999)

[KAS 04] Kasper, R., *Mechanical structures in mechatronic systems*, Proceedings of NAFEMS Seminar: "Mechatronics in Structural Analysis", Wiesbaden (2004)

[KEC 97] Kecskemethy, A., Krupp, T., and Hiller, M., *Symbolic processing of multi-loop mechanism dynamics using closed form kinematic solutions*, Multibody System Dynamics I, no. 1, pp. 23-45 (1997)

[KUN 97] P. Kunkel, V. Mehrmann, W. Rath, J. Weickert, *A new software package for linear differential-algebraic equations,* SIAM J. Sci Comput. 18, pp. 115-138 (1997),

[LUB 92] Ch. Lubich, U. Nowak, U. Pohle and Ch. Engstler, *MEXX - Numerical Software for the Integration of Constrained Mechanical Multibody Systems*, Preprint SC 92-12 (December 1992)

[MML 95] McMillan, S. and Orin, D., *Efficient computation of articulated-body inertias using successive axial screws IEEE Trans. Robotics and Automation* 11, pp. 606-611 (1995)

[NIK 82] P. E. Nikravesh and I. S. Chung, *Application of Euler parameters to the dynamic analysis of three-dimensional constrained mechanical systems*, J. Mechanical Design, Vol. 104, pp. 785-791 (1982)

[PRE 02] W. H. Press, S. A. Teukolsky, et al., *Numerical Recipes in C++ – The Art of Scientific Computing*, Cambridge, MA, Cambridge University Press, (2002)

[ROO 00] Roosta, S. H., 2000, *Parallel Processing and Parallel Algorithms: Theory and Computation,* Springer, New York. Sachdev, C., "Cooperative robots share the load," Tech. rep., TRN News, URL http://www. trnmag. com/Stories/2002/021302/Cooperative_robots_share_the_load_021302.html (2002)

[SHA 89] A. A. Shabana, *Dynamics of Multibody Systems*, John Wiley and Sons, New York (1989)

[SHA 01] Shabana, A. A., *Computational dynamics,* Wiley, New York (2001)

[SHL 90] W. Schiehlen (Editor): *Multibody Systems Handbook*, Springer (1990)

[CHL 90a] Schiehlen, W., *Multibody systems and robot dynamics,* Proc. SthCISM-IFToMM Symposium on Theory and Practice of Robot Manipulators (A. Morecki, G. Bianchi, and K. Jaworek, eds.), Warsaw, Poland, pp. 14-21 (1990)

[SHL 93] W. Schiehlen (Editor): *Advanced Multiboudy Systems Dymanics, Simulation and Software Tools*, Kluwer Academic Publishers (1993)

[STE 96] Stejskal, V. and Valasek, M., *Kinematics and Dynamics of Machinery*, Marcel Dekker, New York (1996)

[STE 01] Stejskal, V.; Dehombreux, P.; Eiber, A.; Gupta, R.; Okrouhlik, M., *Mechanics with Matlab. EC Project Leonardo da Vinci "MechMat" between Universities of Prague, Mons, Stuttgart, Uppsala and the Czech Academy of Science,* Electronic Internet Publication, http://www.fsid.cvut.cz/cp1250/en/U2052/leo.html

[SWB 02] Schwab, A. L. *Dynamics of Flexible Multibody Systems*, PhD thesis, Delft University of Technology. pp. 100-155 (2002)

[VER 74] Vereshchagin, A., *Computer simulation of the dynamics of complicated mechanisms of robot manipulators*, *Engineering Cybernetics* 6, pp. 65-70 (1974)

[WAL 82] Walker, M. and Orin, D., *Efficient dynamic computer simulation of robotic mechanisms*, ASME Journal of Dynamic Systems, Measurement and Control 104, pp. 205-211 (1982)

[WAN 00] Wang, J., Gosselin, C., and Cheng, L., *Dynamic modelling and simulation of parallel mechanisms using virtual spring approach*, Proc. 2000 ASME Design Engineering Technical Conferences, Baltimore, Maryland, pp. 1-10 (2000).

[WIT 77] Wittenburg, J., *Dynamics of systems of rigid bodies,* B. G. Teubner, Stuttgart (1977)

[ZOY 93] Zoyama, A., *Modelling and Simulation of Robot Manipulators: A Parallel Processing Approach*, World Scientific, River Edge, NJ (1993)